ooRexx Documentation 4.2.4

ooDialog Reference

Friday, December 23, 2022 svn revision 12583

ooRexx Documentation 4.2.4 ooDialog Reference
Friday, December 23, 2022 svn revision 12583
Edition 2022.12.22

Author	Open Object Rexx™
Author	W. David Ashley
Author	Rony G. Flatscher
Author	Rick McGuire
Author	Mark Miesfeld
Author	Lee Peedin
Author	Oliver Sims
Author	Erich Steinböck
Author	Jon Wolfers

Preface **liii**
1. How This Book is Structured ... liii
2. Document Conventions ... liii
2.1. Typographic Conventions ... liii
2.2. Notes and Warnings ... liv
3. How to Read the Syntax Diagrams ... liv
4. Getting Help and Submitting Feedback ... lvi
4.1. The Open Object Rexx SourceForge Site ... lvii
4.2. The Rexx Language Association Mailing List ... lviii
4.3. comp.lang.rexx Newsgroup ... lviii
5. Related Information ... lviii

1. Brief Overview **1**
1.1. Getting Started ... 1
1.2. Definition of Terms ... 1
1.2.1. Client / Nonclient Area ... 1
1.2.2. Color ... 1
1.2.3. Color Palette ... 2
1.2.4. COLORREF ... 2
1.2.5. Conventional Hexadecimal Format ... 2
1.2.6. #define Statement ... 3
1.2.7. Device Context ... 3
1.2.8. Dialog Icon ... 3
1.2.9. Dialog Unit ... 3
1.2.10. Handle ... 4
1.2.11. Header File ... 4
1.2.12. Modal and Modeless Dialogs ... 4
1.2.13. Pixel ... 5
1.2.14. Resource Editor ... 5
1.2.15. Resource ID ... 5
1.2.16. Resource Script ... 5
1.2.17. Screen / Client Coordinates ... 5
1.2.18. Symbolic ID ... 5
1.2.19. SystemColor ... 6
1.2.20. System Error Code ... 8
1.2.21. Windows Documentation ... 8
1.2.22. Windows Platform SDK ... 8
1.3. Common Concepts ... 9
1.3.1. Deprecated ... 9
1.3.2. Dialog Template ... 9
1.3.3. Events ... 9
1.3.4. factorX / factorY ... 10
1.3.5. IBM Resource Workshop ... 10
1.3.6. Numbers in ooDialog: ... 11
1.3.7. Predefined Symbolic IDs ... 11
1.3.8. Required Common Control Library (Comctl32) Version ... 12
1.3.9. Rectangle Coordinates ... 12
1.3.10. Required Windows Version ... 12
1.3.11. Underlying Dialog Creation ... 13
1.3.12. Undocumented Items ... 13
1.3.13. Using Symbolic IDs in ooDialog: ... 13
1.3.14. Window Messages ... 16
1.4. History ... 17
1.4.1. Prehistory Timeline ... 17

1.4.2. ooDialog Release 4.2.0 19
1.4.3. ooDialog Release 4.2.1 24
1.4.4. ooDialog Release 4.2.2 28
1.4.5. ooDialog Release 4.2.3 33
1.5. Current Release 36
1.6. Future 37
1.7. ooDialog Class Reference 38

2. The ooDialog Executable 41

2.1. OverView 41
2.2. ooDialog.Exe Command Line Options 41
2.3. ooDialog.Exe as a Program Launcher 42
2.4. ooDialog.Exe Configure File Associations 45

3. The Dialog Object 56

3.1. Method Table 56
3.2. Constant Methods 68
3.3. Class Methods 69
3.3.1. getFontName (Class method) 70
3.3.2. getFontSize (Class method) 70
3.3.3. new (Class method) 70
3.3.4. setDefaultFont (Class method) 72
3.4. Attribute Methods 73
3.4.1. autoDetect (Attribute) 73
3.4.2. constDir (Attribute) 74
3.4.3. dlgHandle (Attribute) 74
3.4.4. dlgID (Attribute) 74
3.4.5. factorX (Attribute) 76
3.4.6. factorY (Attribute) 76
3.4.7. fontName (Attribute) 76
3.4.8. fontSize (Attribute) 78
3.4.9. hwnd (Attribute) 78
3.4.10. initCode (Attribute) 78
3.4.11. ownerDialog (Attribute) 78
3.4.12. pixelCX (Attribute) 80
3.4.13. pixelCY (Attribute) 80
3.4.14. sizeX (Attribute) 80
3.4.15. sizeY (Attribute) 80
3.5. Basic Window Methods 80
3.5.1. childWindowFromPoint 80
3.5.2. clear 81
3.5.3. client2screen 81
3.5.4. clientRect 81
3.5.5. clientToScreen 81
3.5.6. disable 81
3.5.7. display 81
3.5.8. draw 81
3.5.9. enable 82
3.5.10. foregroundWindow 82
3.5.11. getClientRect 82
3.5.12. getExStyleRaw 82
3.5.13. getID 82
3.5.14. getPos 82
3.5.15. getRealPos 82
3.5.16. getRealSize 83

3.5.17. getRect 83
3.5.18. getSize 83
3.5.19. getStyleRaw 83
3.5.20. getText 83
3.5.21. getTextSizePx 83
3.5.22. getTextSizeScreen 83
3.5.23. hide 84
3.5.24. hideFast 84
3.5.25. isEnabled 84
3.5.26. isVisible 84
3.5.27. mapWindowPoints 84
3.5.28. move 84
3.5.29. moveTo 84
3.5.30. moveWindow 85
3.5.31. redraw 85
3.5.32. redrawClient 85
3.5.33. resize 85
3.5.34. resizeTo 85
3.5.35. screen2client 86
3.5.36. screenToClient 86
3.5.37. sendMessage 86
3.5.38. sendMessageHandle 86
3.5.39. setRect 86
3.5.40. setStyleRaw 87
3.5.41. setText 87
3.5.42. setTitle 87
3.5.43. setWindowPos 87
3.5.44. showFast 88
3.5.45. sizeWindow 88
3.5.46. title 88
3.5.47. title= 88
3.5.48. update 88
3.5.49. updateWindow 88
3.5.50. windowRect 89
3.6. Extended Window Methods 89
3.6.1. createBrush 89
3.6.2. createFont 89
3.6.3. createFontEx 89
3.6.4. createPen 89
3.6.5. deleteFont 90
3.6.6. deleteObject 90
3.6.7. drawAngleArc 90
3.6.8. drawArc 90
3.6.9. drawLine 90
3.6.10. drawPie 90
3.6.11. drawPixel 90
3.6.12. fillDrawing 91
3.6.13. fillRect 91
3.6.14. fontColor 91
3.6.15. fontToDC 91
3.6.16. freeDC 91
3.6.17. getArcDirection 91
3.6.18. getDC 91
3.6.19. getFont 92

3.6.20. getPixel 92
3.6.21. getSysBrush 92
3.6.22. getTextAlign 92
3.6.23. getTextExtent 92
3.6.24. hScrollPos 92
3.6.25. loadBitmap 92
3.6.26. objectToDC 93
3.6.27. opaqueText 93
3.6.28. rectangle 93
3.6.29. removeBitmap 93
3.6.30. scroll 93
3.6.31. setArcDirection 93
3.6.32. setFont 93
3.6.33. setHScrollPos 94
3.6.34. setTextAlign 94
3.6.35. setVScrollPos 94
3.6.36. transparentText 94
3.6.37. vScrollPos 94
3.6.38. write 94
3.6.39. writeDirect 95
3.7. Preparing and Running the Dialog 95
3.7.1. addNewAttribute 95
3.7.2. addNewMethod 96
3.7.3. addAutoStartMethod 97
3.7.4. endAsyncExecution 97
3.7.5. execute 97
3.7.6. executeAsync 99
3.7.7. initDialog 99
3.7.8. isDialogActive 99
3.7.9. parseIncludefile 100
3.7.10. popup 100
3.7.11. popupAsChild 100
3.7.12. resolveNumericID 100
3.7.13. resolveSymbolicID 100
3.7.14. setDlgFont 100
3.8. Instantiating Dialog Controls 101
3.8.1. createToolTip 102
3.8.2. newCheckBox 103
3.8.3. newComboBox 104
3.8.4. newDateTimePicker 104
3.8.5. newEdit 105
3.8.6. newGroupBox 106
3.8.7. newListBox 106
3.8.8. newListView 107
3.8.9. newMonthCalendar 107
3.8.10. newProgressBar 108
3.8.11. newPushButton 109
3.8.12. newRadioButton 109
3.8.13. newReBar 110
3.8.14. newScrollBar 110
3.8.15. newStatic 111
3.8.16. newStatusBar 111
3.8.17. newTab 112
3.8.18. newToolBar 112

3.8.19. newToolTip 113
3.8.20. newTrackBar 114
3.8.21. newTreeView 115
3.8.22. newUpDown 116
3.9. Connecting Event Methods 116
3.9.1. addUserMsg 116
3.9.2. connectActivate 116
3.9.3. connectAllSBEvents 116
3.9.4. connectButtonEvent 117
3.9.5. connectComboBoxEvent 117
3.9.6. connectCommandEvents 117
3.9.7. connectDateTimePickerEvent 117
3.9.8. connectDraw 117
3.9.9. connectEachSBEvent 117
3.9.10. connectEditEvent 118
3.9.11. connectFKeyPress 118
3.9.12. connectHelp 118
3.9.13. connectKeyPress 118
3.9.14. connectListBoxEvent 118
3.9.15. connectListViewEvent 118
3.9.16. connectMonthCalendarEvent 119
3.9.17. connectMove 119
3.9.18. connectNotifyEvent 119
3.9.19. connectPosChanged 119
3.9.20. connectResize 119
3.9.21. connectResizing 119
3.9.22. connectScrollBarEvent 119
3.9.23. connectSizeMoveEnded 120
3.9.24. connectStaticEvent 120
3.9.25. connectTabEvent 120
3.9.26. connectToolBarEvent 120
3.9.27. connectToolTipEvent 120
3.9.28. connectTrackBarEvent 120
3.9.29. connectTreeViewEvent 121
3.9.30. connectUpDownEvent 121
3.9.31. defListDragHandler 121
3.9.32. defTreeDragHandler 121
3.9.33. disconnectKeyPress 121
3.9.34. hasKeyPressConnection 121
3.10. Standard Event Handling Methods 121
3.10.1. cancel 122
3.10.2. help 122
3.10.3. ok 123
3.10.4. leaving 124
3.10.5. validate 124
3.11. Appearance and Behavior Methods 125
3.11.1. backgroundColor 125
3.11.2. backgroundSysColor 125
3.11.3. center 126
3.11.4. disableControl 126
3.11.5. enableControl 127
3.11.6. ensureVisible 128
3.11.7. focusControl 128
3.11.8. getControlRect 129

3.11.9. getControlText ... 129
3.11.10. getFocus ... 129
3.11.11. getWindowText ... 130
3.11.12. hideControl ... 131
3.11.13. hideControlFast ... 131
3.11.14. hideWindow ... 132
3.11.15. hideWindowFast ... 133
3.11.16. isMaximized ... 133
3.11.17. isMinimized ... 133
3.11.18. maximize ... 134
3.11.19. minimize ... 135
3.11.20. moveControl ... 135
3.11.21. resizeControl ... 136
3.11.22. restore ... 136
3.11.23. setControlColor ... 136
3.11.24. setControlFont ... 136
3.11.25. setControlSysColor ... 137
3.11.26. setControlText ... 137
3.11.27. setFocus ... 137
3.11.28. setFocusToWindow ... 138
3.11.29. setForegroundWindow ... 138
3.11.30. setGroup ... 139
3.11.31. setTabStop ... 139
3.11.32. setWindowRect ... 140
3.11.33. setWindowText ... 141
3.11.34. show ... 141
3.11.35. showControl ... 142
3.11.36. showControlFast ... 143
3.11.37. showWindow ... 143
3.11.38. showWindowFast ... 143
3.11.39. tabToNext ... 143
3.11.40. tabToPrevious ... 144
3.11.41. toTheTop ... 144
3.11.42. FileViewer Example Program ... 145
3.12. Bitmap Methods ... 148
3.12.1. backgroundBitmap ... 148
3.12.2. changeBitmapButton ... 148
3.12.3. dimBitmap ... 148
3.12.4. displaceBitmap ... 148
3.12.5. drawBitmap ... 149
3.12.6. getBitmapPosition ... 149
3.12.7. getBitmapSize ... 149
3.12.8. getBitmapSizeX ... 149
3.12.9. getBitmapSizeY ... 149
3.12.10. getBmpDisplacement ... 149
3.12.11. installAnimatedButton ... 149
3.12.12. installBitmapButton ... 150
3.12.13. scrollBitmapFromTo ... 150
3.12.14. scrollButton ... 150
3.12.15. setBitmapPosition ... 150
3.12.16. tiledBackgroundBitmap ... 150
3.13. Combo Box Methods ... 151
3.13.1. addComboEntry ... 151
3.13.2. changeComboEntry ... 151

3.13.3. comboAddDirectory 152
3.13.4. comboDrop 153
3.13.5. deleteComboEntry 153
3.13.6. findComboEntry 153
3.13.7. getComboEntry 154
3.13.8. getComboItems 154
3.13.9. getCurrentComboIndex 154
3.13.10. insertComboEntry 155
3.13.11. setCurrentComboIndex 155
3.14. Data Attribute Methods 156
3.14.1. Understanding Data Attributes 156
3.14.2. Data Attribute Names 157
3.14.3. Problems with Data Attributes 161
3.14.4. autoDetection 161
3.14.5. Connect Data Attribute Methods 161
3.14.6. getCheckBoxData 176
3.14.7. getComboBoxData 177
3.14.8. getControlData 177
3.14.9. getData 178
3.14.10. getDataAttribute 179
3.14.11. getDataStem 179
3.14.12. getEditData 180
3.14.13. getListBoxData 180
3.14.14. getRadioButtonData 181
3.14.15. initAutoDetection 181
3.14.16. noAutoDetection 182
3.14.17. setCheckBoxData 182
3.14.18. setComboBoxData 183
3.14.19. setControlData 183
3.14.20. setData 184
3.14.21. setDataAttribute 185
3.14.22. setDataStem 186
3.14.23. setEditData 187
3.14.24. setListBoxData 188
3.14.25. setRadioButtonData 189
3.15. List Box Methods 189
3.15.1. addListEntry 190
3.15.2. changeListEntry 190
3.15.3. deleteListEntry 190
3.15.4. findListEntry 191
3.15.5. getCurrentListIndex 191
3.15.6. getListEntry 191
3.15.7. getListItemHeight 192
3.15.8. getListItemHeightPx 192
3.15.9. getListItems 192
3.15.10. getListWidth 192
3.15.11. getListWidthPx 192
3.15.12. insertListEntry 192
3.15.13. listAddDirectory 193
3.15.14. listDrop 193
3.15.15. setCurrentListIndex 193
3.15.16. setListColumnWidth 194
3.15.17. setListColumnWidthPx 194
3.15.18. setListItemHeight 194

3.15.19. setListItemHeightPx 194
3.15.20. setListWidth 194
3.15.21. setListWidthPx 194
3.15.22. setListTabulators 194
3.16. Menu Methods 195
3.16.1. getMenuBar 195
3.16.2. hasMenuBar 196
3.17. Operating System Related Methods 196
3.17.1. dlgUnit2pixel 196
3.17.2. getSelf 197
3.17.3. get 197
3.17.4. getControlHandle 198
3.17.5. getControlID 199
3.17.6. getTextSizeDu 200
3.17.7. getTextSizeDlg 201
3.17.8. getTextSizeTitleBar 202
3.17.9. pixel2dlgUnit 203
3.18. Scroll Bar Methods 204
3.18.1. determineSBPosition 204
3.18.2. getSBPos 204
3.18.3. getSBRange 205
3.18.4. setSBPos 205
3.18.5. setSBRange 205
3.19. Sending Window Messages 205
3.19.1. sendMessageToControl 205
3.19.2. sendMessageToControlH 206
3.19.3. sendMessageToWindow 207
3.19.4. sendMessageToWindowH 208
3.20. Window Draw Methods 208
3.20.1. clearControlRect 209
3.20.2. clearRect 209
3.20.3. clearWindowRect 209
3.20.4. createBrush 209
3.20.5. drawButton 209
3.20.6. freeControlDC 210
3.20.7. freeWindowDC 210
3.20.8. getControlDC 210
3.20.9. getWindowDC 210
3.20.10. redrawControl 210
3.20.11. redrawRect 210
3.20.12. redrawWindow 210
3.20.13. redrawWindowRect 211
3.20.14. scrollInControl 211
3.20.15. scrollText 211
3.20.16. writeToWindow 211
3.20.17. writeToControl 211

4. Base and Mixin Classes 212
4.1. Component Classes Diagram 212
4.2. CreateWindows Mixin Class 213
4.2.1. Method Table 213
4.2.2. createReBarWindow 213
4.2.3. createStatusBarWindow 215
4.2.4. createToolBarWindow 217

4.3. CustomDraw Mixin Class 218
4.3.1. Custom Draw Overview 218
4.3.2. Custom Draw in ooDialog 220
4.3.3. Simple Custom Draw 221
4.3.4. Method Table 221
4.3.5. Constants 221
4.3.6. customDraw 223
4.3.7. customDrawControl 224
4.3.8. rgb 225
4.4. DialogControl 226
4.4.1. Method Table 226
4.4.2. assignFocus 227
4.4.3. clearRect 227
4.4.4. connectCharEvent 228
4.4.5. connectFKeyPress 228
4.4.6. connectKeyPress 228
4.4.7. data 228
4.4.8. data= 228
4.4.9. disconnectKeyPress 229
4.4.10. getTextSizeDlg 229
4.4.11. group 229
4.4.12. hasKeyPressConnection 229
4.4.13. redrawRect 229
4.4.14. setColor 230
4.4.15. setSysColor 230
4.4.16. tabStop 230
4.4.17. textSize 230
4.4.18. useUnicode 230
4.4.19. usingUnicode 230
4.5. DialogExtensions Mixin Class 231
4.5.1. Method Table 231
4.5.2. addAutoStartMethod 233
4.5.3. changeBitmapButton 233
4.5.4. clearControlRect 235
4.5.5. clearRect 236
4.5.6. clearWindowRect 238
4.5.7. createBrush 238
4.5.8. determineSBPosition 239
4.5.9. dimBitmap 240
4.5.10. displaceBitmap 241
4.5.11. drawBitmap 242
4.5.12. drawButton 243
4.5.13. endAsyncExecution 243
4.5.14. executeAsync 243
4.5.15. freeControlDC 245
4.5.16. freeWindowDC 245
4.5.17. getBitmapPosition 246
4.5.18. getBitmapSize 246
4.5.19. getBitmapSizeX 247
4.5.20. getBitmapSizeY 247
4.5.21. getBmpDisplacement 248
4.5.22. getControlDC 248
4.5.23. getControlRect 249
4.5.24. getListItemHeight 249

4.5.25. getListItemHeightPx 249
4.5.26. getListWidth 250
4.5.27. getListWidthPx 250
4.5.28. getSBPos 250
4.5.29. getSBRange 250
4.5.30. getWindowDC 251
4.5.31. installAnimatedButton 251
4.5.32. installBitmapbutton 253
4.5.33. moveControl 254
4.5.34. popup 255
4.5.35. popupAsChild 257
4.5.36. redrawControl 259
4.5.37. redrawRect 259
4.5.38. redrawWindow 261
4.5.39. redrawWindowRect 261
4.5.40. resizeControl 262
4.5.41. scrollBitmapFromTo 262
4.5.42. scrollButton 263
4.5.43. scrollInControl 264
4.5.44. scrollText 266
4.5.45. setBitmapPosition 268
4.5.46. setControlColor 269
4.5.47. setControlFont 270
4.5.48. setControlSysColor 270
4.5.49. setForegroundWindow 271
4.5.50. setListColumnWidth 272
4.5.51. setListColumnWidthPx 273
4.5.52. setListItemHeight 273
4.5.53. setListItemHeightPx 273
4.5.54. setListWidth 274
4.5.55. setListWidthPx 274
4.5.56. setSBPos 275
4.5.57. setSBRange 275
4.5.58. setWindowRect 276
4.5.59. writeToWindow 278
4.5.60. writeToControl 280
4.6. DynamicDialog Mixin Class 282
4.6.1. Method Table 282
4.6.2. addIconResource 284
4.6.3. create 284
4.6.4. createBitmapButton 284
4.6.5. createBlackFrame 285
4.6.6. createBlackRect 285
4.6.7. createCenter 285
4.6.8. createCheckBox 285
4.6.9. createCheckBoxGroup 285
4.6.10. createCheckBoxStem 286
4.6.11. createComboBox 286
4.6.12. createComboBoxInput 286
4.6.13. createDateTimePicker 286
4.6.14. createEdit 286
4.6.15. createEditInput 286
4.6.16. createEditInputGroup 287
4.6.17. createEditInputStem 287

4.6.18. createEtchedFrame ... 287
4.6.19. createEtchedHorizontal ... 287
4.6.20. createEtchedVertical ... 287
4.6.21. createGrayFrame ... 287
4.6.22. createGrayRect ... 288
4.6.23. createGroupbox ... 288
4.6.24. createListBox ... 288
4.6.25. createListView ... 288
4.6.26. createMonthCalendar ... 288
4.6.27. createOkCancelLeftBottom ... 288
4.6.28. createOkCancelLeftTop ... 288
4.6.29. createOkCancelRightBottom ... 289
4.6.30. createOkCancelRightTop ... 289
4.6.31. createPasswordEdit ... 289
4.6.32. createProgressBar ... 289
4.6.33. createPushButton ... 289
4.6.34. createPushButtonGroup ... 289
4.6.35. createRadioButton ... 290
4.6.36. createRadioButtonGroup ... 290
4.6.37. createRadioButtonStem ... 290
4.6.38. createScrollBar ... 290
4.6.39. createStatic ... 290
4.6.40. createStaticFrame ... 290
4.6.41. createStaticImage ... 291
4.6.42. createStaticText ... 291
4.6.43. createWhiteFrame ... 291
4.6.44. createWhiteRect ... 291
4.6.45. createTab ... 291
4.6.46. createToolBar ... 291
4.6.47. createTrackBar ... 292
4.6.48. createTreeView ... 292
4.6.49. createUpDown ... 292
4.6.50. defineDialog ... 292
4.6.51. load ... 292
4.6.52. loadFrame ... 292
4.6.53. loadItems ... 292
4.7. EventNotification Mixin Class ... 293
4.7.1. Connecting Event Handlers ... 293
4.7.2. Coding Event Handlers ... 293
4.7.3. Coding Event Handler Examples ... 295
4.7.4. Common *willReply* argument ... 299
4.7.5. Method Table ... 300
4.7.6. addUserMsg ... 301
4.7.7. connectActivate ... 310
4.7.8. connectButtonEvent ... 312
4.7.9. connectComboBoxEvent ... 317
4.7.10. connectCommandEvents ... 320
4.7.11. connectDateTimePickerEvent ... 324
4.7.12. connectDraw ... 337
4.7.13. connectEditEvent ... 342
4.7.14. connectHelp ... 345
4.7.15. Connecting Key Press Events ... 348
4.7.16. connectListBoxEvent ... 358
4.7.17. connectListViewEvent ... 360

4.7.18. connectMonthCalendarEvent ... 381
4.7.19. connectMove ... 386
4.7.20. connectNotifyEvent ... 388
4.7.21. connectPosChanged ... 391
4.7.22. connectReBarEvent ... 392
4.7.23. connectResize ... 397
4.7.24. connectResizing ... 399
4.7.25. Connecting Scroll Bar Events ... 404
4.7.26. connectSizeMoveEnded ... 410
4.7.27. connectStaticEvent ... 412
4.7.28. connectStatusBarEvent ... 415
4.7.29. connectTabEvent ... 419
4.7.30. connectToolBarEvent ... 423
4.7.31. connectToolTipEvent ... 428
4.7.32. connectTrackBarEvent ... 433
4.7.33. connectTreeViewEvent ... 436
4.7.34. connectUpDownEvent ... 457
4.7.35. defListDragHandler ... 459
4.7.36. defTreeDragHandler ... 460
4.8. PlainBaseDialog Class ... 461
4.8.1. Method Table ... 461
4.8.2. getFontName (Class method) ... 467
4.8.3. getFontSize (Class method) ... 467
4.8.4. new (Class method) ... 467
4.8.5. setDefaultFont (Class method) ... 467
4.8.6. autoDetect (Attribute) ... 467
4.8.7. dlgHandle (Attribute) ... 467
4.8.8. fontName (Attribute) ... 467
4.8.9. fontSize (Attribute) ... 468
4.8.10. ownerDialog (Attribute) ... 468
4.8.11. addComboEntry ... 468
4.8.12. addListEntry ... 468
4.8.13. addNewAttribute ... 468
4.8.14. addNewMethod ... 468
4.8.15. backgroundBitmap ... 468
4.8.16. autoDetection ... 469
4.8.17. backgroundColor ... 469
4.8.18. cancel ... 469
4.8.19. center ... 469
4.8.20. changeComboEntry ... 469
4.8.21. changeListEntry ... 469
4.8.22. comboAddDirectory ... 470
4.8.23. comboDrop ... 470
4.8.24. connectCheckBox ... 470
4.8.25. connectComboBox ... 470
4.8.26. connectDateTimePicker ... 470
4.8.27. connectEdit ... 470
4.8.28. connectListBox ... 470
4.8.29. connectListView ... 471
4.8.30. connectMonthCalendar ... 471
4.8.31. connectRadioButton ... 471
4.8.32. connectTab ... 471
4.8.33. connectTrackBar ... 471
4.8.34. connectTreeview ... 471

4.8.35. connectUpDown 472
4.8.36. deleteComboEntry 472
4.8.37. deleteListEntry 472
4.8.38. disableControl 472
4.8.39. dlgUnit2pixel 472
4.8.40. enableControl 472
4.8.41. ensureVisible 472
4.8.42. execute 473
4.8.43. findComboEntry 473
4.8.44. findListEntry 473
4.8.45. focusControl 473
4.8.46. get 473
4.8.47. getCheckBoxData 473
4.8.48. getComboBoxData 473
4.8.49. getComboEntry 474
4.8.50. getComboItems 474
4.8.51. getControlData 474
4.8.52. getControlHandle 474
4.8.53. getControlID 474
4.8.54. getControlText 474
4.8.55. getCurrentComboIndex 474
4.8.56. getCurrentListIndex 475
4.8.57. getData 475
4.8.58. getDataAttribute 475
4.8.59. getDataStem 475
4.8.60. getEditData 475
4.8.61. getFocus 475
4.8.62. getListBoxData 475
4.8.63. getListEntry 476
4.8.64. getListItems 476
4.8.65. getMenuBar 476
4.8.66. getRadioButtonData 476
4.8.67. getSelf 476
4.8.68. getTextSizeDlg 476
4.8.69. getTextSizeDu 476
4.8.70. getWindowText 477
4.8.71. hasMenuBar 477
4.8.72. help 477
4.8.73. HideControl 477
4.8.74. hideControlFast 477
4.8.75. hideWindow 477
4.8.76. hideWindowFast 477
4.8.77. initAutoDetection 478
4.8.78. initDialog 478
4.8.79. insertComboEntry 478
4.8.80. insertListEntry 478
4.8.81. isDialogActive 478
4.8.82. isMaximized 478
4.8.83. isMinimized 478
4.8.84. leaving 479
4.8.85. listAddDirectory 479
4.8.86. listDrop 479
4.8.87. maximize 479
4.8.88. minimize 479

4.8.89. newCheckBox 479
4.8.90. newComboBox 479
4.8.91. newDateTimePicker 480
4.8.92. newEdit 480
4.8.93. newGroupBox 480
4.8.94. newListBox 480
4.8.95. newListView 480
4.8.96. newMonthCalendar 480
4.8.97. newProgressBar 480
4.8.98. newPushButton 481
4.8.99. newRadioButton 481
4.8.100. newScrollBar 481
4.8.101. newStatic 481
4.8.102. newTab 481
4.8.103. newToolBar 481
4.8.104. newToolTip 481
4.8.105. newTrackBar 482
4.8.106. newTreeView 482
4.8.107. newUpDown 482
4.8.108. noAutoDetection 482
4.8.109. ok 482
4.8.110. pixel2dlgUnit 482
4.8.111. restore 482
4.8.112. sendMessageToControl 483
4.8.113. sendMessageToControlH 483
4.8.114. sendMessageToWindow 483
4.8.115. sendMessageToWindowH 483
4.8.116. setCheckBoxData 483
4.8.117. setComboBoxData 483
4.8.118. setControlData 483
4.8.119. setControlText 484
4.8.120. setCurrentComboIndex 484
4.8.121. setCurrentListIndex 484
4.8.122. setData 484
4.8.123. setDataAttribute 484
4.8.124. setDataStem 484
4.8.125. setDlgFont 484
4.8.126. setEditData 485
4.8.127. setFocus 485
4.8.128. setFocusToWindow 485
4.8.129. setGroup 485
4.8.130. setListBoxData 485
4.8.131. setListTabulators 485
4.8.132. setRadioButtonData 486
4.8.133. setTabStop 486
4.8.134. setWindowText 486
4.8.135. show 486
4.8.136. showControl 486
4.8.137. showControlFast 486
4.8.138. showWindow 486
4.8.139. showWindowFast 487
4.8.140. tabToNext 487
4.8.141. tabToPrevious 487
4.8.142. tiledBackgroundBitmap 487

4.8.143. toTheTop ... 487
4.8.144. validate ... 487
4.9. ResourceUtils Mixin Class ... 487
4.9.1. Method Table ... 488
4.9.2. constDir (Attribute) ... 488
4.9.3. parseIncludeFile ... 489
4.9.4. resolveNumericID ... 489
4.9.5. resolveSymbolicID ... 490
4.10. WindowBase Mixin Class ... 491
4.10.1. Method Table ... 491
4.10.2. factorX (Attribute) ... 493
4.10.3. factorY (Attribute) ... 493
4.10.4. hwnd (Attribute) ... 493
4.10.5. initCode (Attribute) ... 494
4.10.6. pixelCX (Attribute) ... 494
4.10.7. pixelCY (Attribute) ... 495
4.10.8. sizeX (Attribute) ... 496
4.10.9. sizeY (Attribute) ... 497
4.10.10. childWindowFromPoint ... 497
4.10.11. clear ... 498
4.10.12. client2screen ... 499
4.10.13. clientRect ... 499
4.10.14. clientToScreen ... 500
4.10.15. disable ... 500
4.10.16. display ... 501
4.10.17. draw ... 502
4.10.18. enable ... 503
4.10.19. foregroundWindow ... 503
4.10.20. getClientRect ... 504
4.10.21. getExStyleRaw ... 504
4.10.22. getID ... 505
4.10.23. getPos ... 505
4.10.24. getRealPos ... 506
4.10.25. getRealSize ... 506
4.10.26. getRect ... 507
4.10.27. getSize ... 507
4.10.28. getStyleRaw ... 508
4.10.29. getText ... 509
4.10.30. getTextSizePx ... 509
4.10.31. getTextSizeScreen ... 510
4.10.32. hide ... 511
4.10.33. hideFast ... 512
4.10.34. isEnabled ... 512
4.10.35. isVisible ... 513
4.10.36. mapWindowPoints ... 513
4.10.37. move ... 514
4.10.38. moveTo ... 515
4.10.39. moveWindow ... 517
4.10.40. redraw ... 519
4.10.41. redrawClient ... 520
4.10.42. resize ... 520
4.10.43. resizeTo ... 522
4.10.44. screen2client ... 523
4.10.45. screenToClient ... 524

4.10.46. sendMessage 524
4.10.47. sendMessageHandle 525
4.10.48. setRect 526
4.10.49. setStyleRaw 528
4.10.50. setText 529
4.10.51. setTitle 530
4.10.52. setWindowPos 530
4.10.53. show 533
4.10.54. showFast 534
4.10.55. sizeWindow 534
4.10.56. title 537
4.10.57. title= 537
4.10.58. update 537
4.10.59. updateWindow 538
4.10.60. windowRect 539
4.11. WindowExtensions Mixin Class 539
4.11.1. Method Table 540
4.11.2. createBrush 541
4.11.3. createFont 542
4.11.4. createFontEx 543
4.11.5. createPen 550
4.11.6. deleteFont 551
4.11.7. deleteObject 551
4.11.8. drawAngleArc 551
4.11.9. drawArc 552
4.11.10. drawLine 553
4.11.11. drawPie 553
4.11.12. drawPixel 553
4.11.13. fillDrawing 554
4.11.14. fillRect 555
4.11.15. fontColor 556
4.11.16. fontToDC 556
4.11.17. freeDC 557
4.11.18. getArcDirection 557
4.11.19. getDC 557
4.11.20. getFont 558
4.11.21. getPixel 558
4.11.22. getSysBrush 559
4.11.23. getTextAlign 559
4.11.24. getTextExtent 560
4.11.25. hScrollPos 561
4.11.26. loadBitmap 561
4.11.27. objectToDC 562
4.11.28. opaqueText 562
4.11.29. rectangle 563
4.11.30. removeBitmap 563
4.11.31. scroll 564
4.11.32. setArcDirection 564
4.11.33. setFont 565
4.11.34. setHScrollPos 565
4.11.35. setTextAlign 566
4.11.36. setVScrollPos 567
4.11.37. transparentText 567
4.11.38. vScrollPos 567

4.11.39. write 567
4.11.40. writeDirect 570

5. Property Sheet and Control Dialogs 571

5.1. Control Dialogs 571
5.1.1. ControlDialog Class 571
5.1.2. RcControlDialog Class 574
5.1.3. ResControlDialog Class 577
5.1.4. UserControlDialog Class 578
5.2. PropertySheetDialog Class 580
5.2.1. Method Table 580
5.2.2. new (Class method) 583
5.2.3. appIcon (Attribute) 585
5.2.4. caption (Attribute) 586
5.2.5. header (Attribute) 586
5.2.6. imageList (Attribute) 587
5.2.7. pages (Attribute) 588
5.2.8. resources (Attribute) 588
5.2.9. startPage (Attribute) 589
5.2.10. watermark (Attribute) 589
5.2.11. addPage 590
5.2.12. apply 591
5.2.13. cancelToClose 591
5.2.14. changed 591
5.2.15. enableWizButtons 592
5.2.16. execute 593
5.2.17. getCurrentPageHwnd 593
5.2.18. getPage 594
5.2.19. getResult 594
5.2.20. getTabControl 595
5.2.21. hwndToIndex 595
5.2.22. idToIndex 595
5.2.23. indexToHwnd 595
5.2.24. indexToID 596
5.2.25. indexToPage 596
5.2.26. initDialog 597
5.2.27. insertPage 597
5.2.28. pageToIndex 598
5.2.29. popup 598
5.2.30. popupAsChild 599
5.2.31. pressButton 600
5.2.32. querySiblings 600
5.2.33. rebootSystem 601
5.2.34. recalcPageSizes 601
5.2.35. restartWindows 602
5.2.36. removePage 602
5.2.37. setButtonText 603
5.2.38. setCurSel 604
5.2.39. setCurSelByID 604
5.2.40. setFinishText 605
5.2.41. setHeaderSubTitle 605
5.2.42. setHeaderTitle 605
5.2.43. setNextText 606
5.2.44. setTitle 606

5.2.45. setWizButtons ... 607
5.2.46. showWizButtons ... 608
5.2.47. unchanged ... 609
5.3. Property Sheet Pages ... 609
5.3.1. PropertySheetPage Class ... 610
5.3.2. RcPSPDialog Class ... 637
5.3.3. ResPSPDialog Class ... 640
5.3.4. UserPSPDialog Class ... 643

6. Resizable Dialogs **647**
6.1. DlgArea Class ... 647
6.1.1. Method Table ... 648
6.1.2. init ... 649
6.1.3. t (Attribute) ... 650
6.1.4. l (Attribute) ... 650
6.1.5. b (Attribute) ... 650
6.1.6. r (Attribute) ... 650
6.1.7. top (Attribute) ... 650
6.1.8. left (Attribute) ... 650
6.1.9. width (Attribute) ... 650
6.1.10. height (Attribute) ... 651
6.1.11. margin (Attribute) ... 651
6.1.12. cx ... 651
6.1.13. cy ... 651
6.1.14. h ... 652
6.1.15. hr ... 653
6.1.16. right ... 653
6.1.17. w ... 653
6.1.18. wr ... 654
6.1.19. x ... 654
6.1.20. y ... 654
6.1.21. DlgArea Example ... 655
6.2. DlgAreaU Class ... 656
6.2.1. Method Table ... 657
6.2.2. new (Class method) ... 657
6.2.3. correctionFactor (Attribute) ... 658
6.2.4. lastError (Attribute) ... 658
6.2.5. noMove (Attribute) ... 658
6.2.6. noResize (Attribute) ... 658
6.2.7. updateOnResize (Attribute) ... 659
6.2.8. noMovePut ... 659
6.2.9. noResizePut ... 659
6.2.10. resize ... 660
6.2.11. Creating resizable Dialogs ... 661
6.2.12. Possible Problems ... 662
6.2.13. Sample Code ... 662
6.3. ResizingAdmin Mixin Class ... 663
6.3.1. Understanding How Resizing Works ... 664
6.3.2. Method Table ... 667
6.3.3. maxSize (Attribute) ... 668
6.3.4. minSize (Attribute) ... 669
6.3.5. controlBottom ... 669
6.3.6. controlLeft ... 670
6.3.7. controlRight ... 671

6.3.8. controlSizing ... 672
6.3.9. controlTop ... 673
6.3.10. defaultBottom ... 674
6.3.11. defaultLeft ... 675
6.3.12. defaultRight ... 676
6.3.13. defaultSizing ... 677
6.3.14. defaultTop ... 678
6.3.15. defineSizing ... 679
6.3.16. noMaxSize ... 680
6.3.17. noMinSize ... 681
6.3.18. pagedTab ... 681
6.3.19. useDefaultSizing ... 682
6.3.20. wantSizeEnded ... 683

7. Resource File Dialogs **685**

7.1. ResDialog Class ... 685
7.1.1. new (Class method) ... 685
7.2. RcDialog Class ... 686
7.2.1. new (Class method) ... 687

8. UserDialog Class **690**

8.1. Method Table ... 690
8.2. new (Class method) ... 692
8.3. addIconResource ... 693
8.4. create ... 694
8.5. createCenter ... 697
8.6. Create... Methods ... 699
8.6.1. Understanding the Create Methods ... 699
8.6.2. Common Control Styles ... 702
8.6.3. Create Button Controls ... 703
8.6.4. Create Combo Box Controls ... 735
8.6.5. createDateTimePicker ... 740
8.6.6. Create Edit Controls ... 742
8.6.7. createListBox ... 757
8.6.8. createListView ... 760
8.6.9. createMonthCalendar ... 762
8.6.10. createProgressBar ... 764
8.6.11. createReBar ... 765
8.6.12. createScrollBar ... 768
8.6.13. Create Static Controls ... 769
8.6.14. createStatusBar ... 791
8.6.15. createTab ... 793
8.6.16. createToolBar ... 795
8.6.17. createTrackbar ... 797
8.6.18. createTreeView ... 799
8.6.19. createUpDown ... 802
8.7. defineDialog ... 804
8.8. load ... 804
8.9. loadFrame ... 806
8.10. loadItems ... 807

9. The Dialog Control Object **809**

9.1. Method Table ... 809
9.2. Attribute Methods ... 813
9.2.1. factorX (Attribute) ... 813

9.2.2. factorY (Attribute) 814
9.2.3. hDlg (Attribute) 814
9.2.4. hwnd 814
9.2.5. id (Attribute) 814
9.2.6. initCode (Attribute) 815
9.2.7. oDlg (Attribute) 815
9.2.8. sizeX (Attribute) 815
9.2.9. sizeY (Attribute) 815
9.2.10. pixelCX (Attribute) 815
9.2.11. pixelCY (Attribute) 816
9.3. Basic Window Methods 816
9.3.1. childWindowFromPoint 816
9.3.2. clear 816
9.3.3. client2screen 816
9.3.4. clientRect 816
9.3.5. clientToScreen 816
9.3.6. disable 817
9.3.7. display 817
9.3.8. draw 817
9.3.9. enable 817
9.3.10. foregroundWindow 817
9.3.11. getClientRect 817
9.3.12. getExStyleRaw 817
9.3.13. getID 818
9.3.14. getPos 818
9.3.15. getRealPos 818
9.3.16. getRealSize 818
9.3.17. getRect 818
9.3.18. getSize 818
9.3.19. getStyleRaw 818
9.3.20. getText 819
9.3.21. getTextSizePx 819
9.3.22. getTextSizeScreen 819
9.3.23. hide 819
9.3.24. hideFast 819
9.3.25. isEnabled 819
9.3.26. isVisible 819
9.3.27. mapWindowPoints 820
9.3.28. move 820
9.3.29. moveTo 820
9.3.30. moveWindow 820
9.3.31. redraw 820
9.3.32. redrawClient 821
9.3.33. resize 821
9.3.34. resizeTo 821
9.3.35. screen2client 821
9.3.36. screenToClient 821
9.3.37. sendMessage 821
9.3.38. sendMessageHandle 822
9.3.39. setRect 822
9.3.40. setStyleRaw 822
9.3.41. setText 822
9.3.42. setTitle 822
9.3.43. setWindowPos 823

9.3.44. show 823
9.3.45. showFast 823
9.3.46. sizeWindow 823
9.3.47. title 824
9.3.48. title= 824
9.3.49. update 824
9.3.50. updateWindow 824
9.3.51. windowRect 824
9.4. Extended Window Methods 824
9.4.1. createBrush 825
9.4.2. createFont 825
9.4.3. createFontEx 825
9.4.4. createPen 825
9.4.5. deleteFont 825
9.4.6. deleteObject 825
9.4.7. drawAngleArc 826
9.4.8. drawArc 826
9.4.9. drawLine 826
9.4.10. drawPie 826
9.4.11. drawPixel 826
9.4.12. fillDrawing 826
9.4.13. fillRect 826
9.4.14. freeDC 827
9.4.15. fontColor 827
9.4.16. fontToDC 827
9.4.17. getArcDirection 827
9.4.18. getDC 827
9.4.19. getFont 827
9.4.20. getPixel 827
9.4.21. getSysBrush 828
9.4.22. getTextAlign 828
9.4.23. getTextExtent 828
9.4.24. hScrollPos 828
9.4.25. loadBitmap 828
9.4.26. objectToDC 828
9.4.27. opaqueText 828
9.4.28. rectangle 829
9.4.29. removeBitmap 829
9.4.30. scroll 829
9.4.31. setArcDirection 829
9.4.32. setFont 829
9.4.33. setHScrollPos 829
9.4.34. setTextAlign 830
9.4.35. setVScrollPos 830
9.4.36. transparentText 830
9.4.37. vScrollPos 830
9.4.38. write 830
9.4.39. writeDirect 830
9.5. assignFocus 830
9.6. clearRect 831
9.7. connectCharEvent 832
9.7.1. Char Event Handler 833
9.8. connectFKeyPress 835
9.9. connectKeyPress 837

9.10. data ... 841
9.11. data= ... 842
9.12. disconnectKeyPress ... 843
9.13. getTextSizeDlg ... 844
9.14. group ... 845
9.15. hasKeyPressConnection ... 846
9.16. redrawRect ... 847
9.17. setColor ... 847
9.18. setParent ... 848
9.19. setSysColor ... 850
9.20. setWindowTheme ... 850
9.21. tabStop ... 851
9.22. textSize ... 852
9.23. useUnicode ... 853
9.24. useVersion ... 853
9.25. usingUnicode ... 854
9.26. usingVersion ... 854

10. Button Controls 856

10.1. Animated Buttons ... 856
10.2. Check Boxes ... 859
10.2.1. Method Table ... 860
10.2.2. newCheckBox (dialog object method) ... 860
10.2.3. createCheckBox (UserDialog method) ... 860
10.2.4. getCheckState ... 861
10.2.5. isIndeterminate ... 861
10.2.6. setIndeterminate ... 862
10.3. Group Boxes ... 862
10.3.1. Method Table ... 862
10.3.2. newGroupBox (dialog object method) ... 863
10.3.3. createGroupBox (UserDialog method) ... 863
10.3.4. style= ... 863
10.4. Radio Buttons ... 863
10.4.1. Method Table ... 864
10.4.2. newRadioButton (dialog object method) ... 864
10.4.3. createRadioButton (UserDialog method) ... 864
10.4.4. checkInGroup (Class) ... 865
10.4.5. check ... 865
10.4.6. checked ... 866
10.4.7. getCheckState ... 866
10.4.8. uncheck ... 867
10.5. Push Buttons ... 867
10.5.1. Method Table ... 868
10.5.2. newPushButton (dialog object method) ... 868
10.5.3. createPushButton (UserDialog method) ... 868
10.5.4. connectButtonEvent (dialog object method) ... 868
10.5.5. click ... 869
10.5.6. getIdealSize ... 869
10.5.7. getImage ... 870
10.5.8. getImageList ... 870
10.5.9. getTextMargin ... 871
10.5.10. push ... 872
10.5.11. setImage ... 873
10.5.12. setImageList ... 873

10.5.13. setTextMargin 876
10.5.14. state 876
10.5.15. state= 877
10.5.16. style= 878

11. Combo Box Controls **882**
11.1. Method Table 882
11.2. newComboBox (dialog object method) 884
11.3. createComboBox (UserDialog method) 884
11.4. connectComboBoxEvent (dialog object method) 884
11.5. add 884
11.6. addDirectory 885
11.7. closeDropDown 886
11.8. delete 886
11.9. deleteAll 887
11.10. find 887
11.11. getCue 888
11.12. getCombBoxInfo 889
11.13. getEditControl 890
11.14. getFirstVisible 891
11.15. getHorizontalExtent 891
11.16. getItemHeight 892
11.17. getMinVisible 892
11.18. getText 893
11.19. insert 893
11.20. isDropDown 894
11.21. isDropDownList 894
11.22. isDropDownOpen 895
11.23. isGrandchild 895
11.24. isSimple 896
11.25. items 897
11.26. modify 897
11.27. openDropDown 898
11.28. removeFullColor 898
11.29. select 899
11.30. selected 899
11.31. selectedIndex 900
11.32. selectIndex 900
11.33. setCue 900
11.34. setEditSelection 901
11.35. setFullColor 902
11.36. setHorizontalExtent 903
11.37. setItemHeight 904
11.38. setMinVisible 905

12. Date Time Picker Controls **907**
12.1. Format Strings 907
12.2. Callback Fields 909
12.3. Method Table 909
12.4. newDateTimePicker (dialog object method) 910
12.5. createDateTimePicker (UserDialog method) 910
12.6. connectDateTimePickerEvent (dialog object method) 910
12.7. closeMonthCal 911
12.8. getDateTime 911
12.9. getIdealSize 912

12.10. getInfo 912
12.11. getMonthCal 914
12.12. getMonthCalColor 915
12.13. getMonthCalFont 916
12.14. getMonthCalStyle 917
12.15. getRange 918
12.16. setDateTime 920
12.17. setFormat 920
12.18. setMonthCalColor 921
12.19. setMonthCalFont 922
12.20. setMonthCalStyle 922
12.21. setRange 925

13. Edit Controls **926**

13.1. Method Table 926
13.2. newEdit (dialog object method) 928
13.3. createEdit (UserDialog method) 928
13.4. connectEditEvent (dialog object method) 928
13.5. passwordChar (Attribute) 928
13.6. addStyle 929
13.7. canUndo 930
13.8. clearText 930
13.9. copyText 931
13.10. cutText 931
13.11. disableInternalResize 931
13.12. ensureCaretVisibility 932
13.13. firstVisibleLine 933
13.14. getCue 933
13.15. getLine 934
13.16. getMargins 935
13.17. getRect 935
13.18. getStyle 936
13.19. hideBalloon 937
13.20. isGrandchild 937
13.21. isModified 939
13.22. isSingleLine 939
13.23. lineFromIndex 940
13.24. lineIndex 940
13.25. lineLength 941
13.26. lines 943
13.27. lineScroll 943
13.28. margins 944
13.29. noContextMenu 944
13.30. pasteText 945
13.31. removeStyle 945
13.32. replaceSelText 946
13.33. replaceStyle 947
13.34. scrollCommand 948
13.35. select 949
13.36. selected 950
13.37. selection 950
13.38. setCue 951
13.39. setLimit 952
13.40. setMargins 952

13.41. setModified 953
13.42. setReadOnly 954
13.43. setRect 954
13.44. setTabStops 955
13.45. showBalloon 956
13.46. tab= 958
13.47. undo 958
13.48. wantReturn 958
13.48.1. wantReturn Event Handler 959

14. List Box Controls 961
14.1. Method Table 961
14.2. newListBox (dialog object method) 962
14.3. createListBox (UserDialog method) 962
14.4. connectListBoxEvent (dialog object method) 963
14.5. add 963
14.6. addDirectory 963
14.7. columnWidth= 964
14.8. columnWidthPx= 964
14.9. delete 964
14.10. deleteAll 965
14.11. deSelectIndex 965
14.12. deSelectRange 965
14.13. find 966
14.14. getFirstVisible 966
14.15. getText 967
14.16. hitTestInfo 967
14.17. insert 968
14.18. isSingleSelection 969
14.19. itemHeight 969
14.20. itemHeight= 969
14.21. itemHeightPx 969
14.22. itemHeightPx= 970
14.23. items 970
14.24. makeFirstVisible 970
14.25. modify 970
14.26. select 971
14.27. selected 971
14.28. selectedIndex 971
14.29. selectedIndexes 971
14.30. selectedItems 972
14.31. selectIndex 972
14.32. selectRange 973
14.33. setTabulators 973
14.34. setWidth 974
14.35. setWidthPx 974
14.36. width 974
14.37. widthPx 975

15. List View Controls 976
15.1. View Styles 976
15.2. Method Table 977
15.3. Constant Methods 981
15.4. newListView (dialog object method) 982
15.5. createListView (UserDialog method) 982

15.6. connectListViewEvent (dialog object method) 982
15.7. add 983
15.8. addExtendedStyle 984
15.9. addFullRow 986
15.10. addRow 987
15.11. addRowFromArray 988
15.12. addStyle 989
15.13. alignLeft 990
15.14. alignTop 990
15.15. arrange 990
15.16. bkColor 990
15.17. bkColor= 991
15.18. check 991
15.19. checkAll 992
15.20. columnInfo 992
15.21. columnWidth 993
15.22. columnWidthPX 993
15.23. delete 994
15.24. deleteAll 994
15.25. deleteColumn 994
15.26. deselect 995
15.27. deselectAll 996
15.28. dropHighlighted 996
15.29. edit 996
15.30. endEdit 997
15.31. ensureVisible 997
15.32. find 997
15.33. findNearestXY 998
15.34. findPartial 998
15.35. firstVisible 999
15.36. fixFullRowColumns 999
15.37. focus 1000
15.38. focused 1000
15.39. getCheck 1001
15.40. getColumnCount 1001
15.41. getColumnInfo 1002
15.42. getColumnOrder 1003
15.43. getColumnText 1004
15.44. getExtendedStyle 1005
15.45. getExtendedStyleRaw 1006
15.46. getFullRow 1007
15.47. getHoverTime 1009
15.48. getImageList 1009
15.49. getItem 1010
15.50. getItemData 1012
15.51. getItemInfo 1013
15.52. getItemPos 1014
15.53. getItemRect 1015
15.54. getSubitem 1016
15.55. getSubitemRect 1017
15.56. getToolTips 1018
15.57. getView 1019
15.58. hasCheckBoxes 1019
15.59. hitTestInfo 1020

15.60. insert 1022
15.61. insertColumn 1023
15.62. insertColumnPX 1025
15.63. insertFullRow 1026
15.64. isChecked 1027
15.65. itemInfo 1027
15.66. itemPos 1028
15.67. items 1029
15.68. itemsPerPage 1029
15.69. itemState 1029
15.70. itemText 1030
15.71. last 1030
15.72. lastSelected 1030
15.73. modify 1031
15.74. modifyColumn 1032
15.75. modifyColumnPX 1032
15.76. modifyFullRow 1033
15.77. modifyItem 1034
15.78. modifySubitem 1035
15.79. next 1036
15.80. nextLeft 1036
15.81. nextRight 1037
15.82. nextSelected 1037
15.83. prepare4nItems 1037
15.84. prependFullRow 1038
15.85. previous 1038
15.86. previousSelected 1039
15.87. redrawItems 1039
15.88. removeExtendedStyle 1040
15.89. removeItemData 1040
15.90. removeStyle 1042
15.91. replaceExtendedStyle 1042
15.92. replaceStyle 1044
15.93. scroll 1045
15.94. select 1046
15.95. selected 1046
15.96. selectedItems 1046
15.97. setColumnOrder 1046
15.98. setColumnWidth 1048
15.99. setColumnWidthPX 1048
15.100. setFullRowText 1049
15.101. setHoverTime 1050
15.102. setImageList 1051
15.103. setItemData 1053
15.104. setItemPos 1054
15.105. setItemState 1055
15.106. setItemText 1056
15.107. setToolTips 1057
15.108. setView 1057
15.109. smallSpacing 1058
15.110. snapToGrid 1058
15.111. sortItems 1059
15.112. spacing 1062
15.113. stringWidth 1062

15.114. stringWidthPX 1062
15.115. textBkColor 1063
15.116. textBkColor= 1063
15.117. textColor 1063
15.118. textColor= 1064
15.119. uncheck 1064
15.120. uncheckAll 1065
15.121. updateItem 1065
15.122. LvCustomDrawSimple Class 1066
    15.122.1. CustomDraw Event Handler 1066
    15.122.2. Method Table 1068
    15.122.3. new (Class Method) 1069
    15.122.4. clrText (Attribute) 1069
    15.122.5. clrTextBk (Attribute) 1069
    15.122.6. drawStage (Attribute) 1070
    15.122.7. font (Attribute) 1071
    15.122.8. id (Attribute) 1072
    15.122.9. item (Attribute) 1072
    15.122.10. itemData (Attribute) 1073
    15.122.11. reply (Attribute) 1074
    15.122.12. subItem (Attribute) 1075
15.123. LvFullRow Class 1075
    15.123.1. Internal Sorting Considerations 1076
    15.123.2. Method Table 1077
    15.123.3. fromArray (Class Method) 1078
    15.123.4. new (Class Method) 1079
    15.123.5. userData (Attribute) 1080
    15.123.6. addSubItem 1081
    15.123.7. insertSubItem 1082
    15.123.8. item 1083
    15.123.9. removeSubItem 1083
    15.123.10. subItem 1084
    15.123.11. subItems 1085
15.124. LvItem Class 1085
    15.124.1. Method Table 1086
    15.124.2. new (Class Method) 1086
    15.124.3. columnFormats (Attribute) 1091
    15.124.4. columns (Attribute) 1092
    15.124.5. groupID (Attribute) 1093
    15.124.6. imageIndex (Attribute) 1093
    15.124.7. indent (Attribute) 1094
    15.124.8. index (Attribute) 1094
    15.124.9. itemData (Attribute) 1095
    15.124.10. itemState (Attribute) 1095
    15.124.11. itemStateMask (Attribute) 1096
    15.124.12. mask (Attribute) 1097
    15.124.13. overlayImageIndex (Attribute) 1099
    15.124.14. stateImageIndex (Attribute) 1100
    15.124.15. text (Attribute) 1100
15.125. LvSubItem Class 1101
    15.125.1. Method Table 1102
    15.125.2. new (Class Method) 1102
    15.125.3. imageIndex (Attribute) 1104
    15.125.4. item (Attribute) 1105

15.125.5. mask (Attribute) 1105
15.125.6. subItem (Attribute) 1106
15.125.7. text (Attribute) 1107

16. Month Calendar Controls 1109
16.1. Method Table 1109
16.2. newMonthCalendar (dialog object method) 1111
16.3. createMonthCalendar (UserDialog method) 1111
16.4. connectMonthCalendarEvent (dialog object method) 1111
16.5. date (Attribute) 1111
16.6. addStyle 1112
16.7. getCalendarBorder 1113
16.8. getCalendarCount 1114
16.9. getCALID 1114
16.10. getColor 1115
16.11. getCurrentView 1116
16.12. getFirstDayOfWeek 1116
16.13. getGridInfo 1117
16.14. getMaxSelection 1120
16.15. getMaxTodayWidth 1120
16.16. getMinRect 1120
16.17. getMonthDelta 1121
16.18. getMonthRange 1122
16.19. getRange 1122
16.20. getSelectionRange 1123
16.21. getStyle 1124
16.22. getToday 1125
16.23. hitTestInfo 1125
16.24. removeStyle 1128
16.25. replaceStyle 1129
16.26. setCalendarBorder 1130
16.27. setCALID 1131
16.28. setColor 1132
16.29. setCurrentView 1133
16.30. setDayState 1134
16.31. setDayStateQuick 1135
16.32. setFirstDayOfWeek 1136
16.33. setMaxSelection 1137
16.34. setMonthDelta 1137
16.35. setRange 1138
16.36. setSelectionRange 1138
16.37. setToday 1139
16.38. sizeRectToMin 1139
16.39. DayState Class 1140
16.39.1. Method Table 1140
16.39.2. new (Class Method) 1140
16.39.3. value 1141
16.40. DayStates Class 1142
16.40.1. Method Table 1142
16.40.2. makeDayStateBuffer (Class method) 1142
16.40.3. new (Class Method) 1144
16.40.4. quickDayStateBuffer (Class method) 1144
16.40.5. endMonth (Attribute) 1145
16.40.6. startMonth (Attribute) 1146

16.40.7. getDayState 1146
16.40.8. getDayStateBuffer 1147
16.40.9. putMonth 1148
16.40.10. putYear 1149

17. Progress Bar Controls 1151

17.1. Method Table 1151
17.2. newProgressBar (dialog object method) 1152
17.3. createProgressBar (UserDialog method) 1152
17.4. backgroundColor 1152
17.5. barColor 1153
17.6. getFullRange 1154
17.7. getPos 1154
17.8. setFullRange 1155
17.9. setMarquee 1156
17.10. setPos 1157
17.11. setStep 1157
17.12. step 1157

18. ReBar Controls 1159

18.1. Method Table 1159
18.2. newReBar (dialog object method) 1159
18.3. createReBar (UserDialog method) 1160
18.4. createReBarWindow (CreateWindows method) 1160
18.5. connectReBarEvent (dialog object method) 1160
18.6. deleteBand 1160
18.7. getBandBorders 1161
18.8. getBandCount 1161
18.9. getBandInfo 1162
18.10. getBandMargins 1162
18.11. getBarHeight 1163
18.12. getBarInfo 1163
18.13. getBkColor 1164
18.14. getColorScheme 1164
18.15. getExtendedStyle 1165
18.16. getImageList 1166
18.17. getPalette 1166
18.18. getRect 1167
18.19. getRowCount 1167
18.20. getRowHeight 1168
18.21. getTextColor 1168
18.22. getToolTips 1169
18.23. hitTestInfo 1169
18.24. idToIndex 1170
18.25. insertBand 1170
18.26. maximizeBand 1171
18.27. minimizeBand 1172
18.28. moveBand 1172
18.29. pushChevron 1173
18.30. setBandInfo 1173
18.31. setBandWidth 1174
18.32. setBarInfo 1174
18.33. setBkColor 1175
18.34. setColorScheme 1175
18.35. setExtendedStyle 1176

18.36. setImageList 1176
18.37. setPalette 1177
18.38. setParent 1178
18.39. setTextColor 1178
18.40. setToolTips 1179
18.41. setWindowTheme 1179
18.42. showBand 1180
18.43. sizeToRect 1180
18.44. ReBarBandInfo Class 1181
18.44.1. Method Table 1181
18.44.2. new (Class Method) 1181
18.44.3. bitmapBack (Attribute) 1182
18.44.4. chevronLocation (Attribute) 1182
18.44.5. chevronState (Attribute) 1183
18.44.6. child (Attribute) 1183
18.44.7. clrBack (Attribute) 1184
18.44.8. clrFore (Attribute) 1184
18.44.9. cx (Attribute) 1185
18.44.10. cxHeader (Attribute) 1185
18.44.11. cxIdeal (Attribute) 1186
18.44.12. cxMinChild (Attribute) 1186
18.44.13. cyChild (Attribute) 1187
18.44.14. cyIntegral (Attribute) 1187
18.44.15. cyMaxChild (Attribute) 1187
18.44.16. cyMinChild (Attribute) 1188
18.44.17. id (Attribute) 1188
18.44.18. imageIndex (Attribute) 1189
18.44.19. itemData (Attribute) 1189
18.44.20. mask (Attribute) 1190
18.44.21. style (Attribute) 1190
18.44.22. text (Attribute) 1191

19. Scroll Bar Controls 1192
19.1. Method Table 1192
19.2. newScrollBar (dialog object method) 1193
19.3. createScrollBar (UserDialog method) 1193
19.4. connectScrollBarEvent (dialog object method) 1193
19.5. connectAllSBEvents (dialog object method) 1193
19.6. connectEachSBEvent (dialog object method) 1193
19.7. Scroll Event Constants 1193
19.8. determinePosition 1194
19.9. position 1195
19.10. range 1195
19.11. setPos 1195
19.12. setRange 1196

20. Static Controls 1197
20.1. Method Table 1197
20.2. createBlackFrame (UserDialog method) 1198
20.3. createBlackRect (UserDialog method) 1198
20.4. createEtchedFrame (UserDialog method) 1198
20.5. createEtchedHorizontal (UserDialog method) 1199
20.6. createEtchedVertical (UserDialog method) 1199
20.7. createGrayFrame (UserDialog method) 1199
20.8. createGrayRect (UserDialog method) 1199

20.9. createStatic (UserDialog method) 1199
20.10. createStaticImage (UserDialog method) 1199
20.11. createStaticText (UserDialog method) 1200
20.12. createWhiteFrame (UserDialog method) 1200
20.13. createWhiteRect (UserDialog method) 1200
20.14. connectStaticEvent (dialog object method) 1200
20.15. newStatic (dialog object method) 1200
20.16. getIcon 1200
20.17. getImage 1201
20.18. setIcon 1202
20.19. setImage 1202

21. StatusBar Controls 1204

21.1. Method Table 1205
21.2. newStatusBar (dialog object method) 1206
21.3. createStatusBar (UserDialog method) 1206
21.4. connectStatusBarEvent (dialog object method) 1206
21.5. createStatusBarWindow (CreateWindows method) 1206
21.6. getBorders 1206
21.7. getIcon 1207
21.8. getParts 1208
21.9. getRect 1209
21.10. getText 1209
21.11. getTextLength 1211
21.12. getTipText 1212
21.13. isSimple 1212
21.14. setBkColor 1213
21.15. setIcon 1214
21.16. setMinHeight 1215
21.17. setParts 1215
21.18. setText 1216
21.19. setTipText 1217
21.20. simple 1218

22. Tab Controls 1220

22.1. Method Table 1220
22.2. newTab (dialog object method) 1221
22.3. createTab (UserDialog method) 1221
22.4. connectTabEvent (dialog object method) 1222
22.5. addFullSeq 1222
22.6. addSequence 1223
22.7. adjustToRectangle 1223
22.8. calcDisplayRect 1224
22.9. calcWindowRect 1225
22.10. delete 1225
22.11. deleteAll 1226
22.12. focus 1226
22.13. focused 1226
22.14. getImageList 1227
22.15. getItemRect 1227
22.16. insert 1227
22.17. itemInfo 1228
22.18. items 1229
22.19. last 1229
22.20. modify 1229

22.21. posRectangle 1230
22.22. requiredWindowSize 1230
22.23. rows 1231
22.24. select 1231
22.25. selected 1232
22.26. selectedIndex 1232
22.27. selectIndex 1232
22.28. setImageList 1233
22.29. setItemSize 1234
22.30. setPadding 1235
22.31. setSize 1235

23. ToolBar Controls 1237

23.1. Method Table 1237
23.2. newToolBar (dialog object method) 1237
23.3. createToolBar (UserDialog method) 1238
23.4. connectToolBarEvent (dialog object method) 1238
23.5. addBitmap 1238
23.6. addButtons 1238
23.7. addString 1239
23.8. autoSize 1240
23.9. buttonCount 1240
23.10. changeBitmap 1241
23.11. checkButton 1241
23.12. commandToIndex 1242
23.13. customize 1242
23.14. getButton 1243
23.15. getButtonText 1243
23.16. getButtonTextEx 1244
23.17. getDisabledImageList 1244
23.18. getExtendedStyle 1245
23.19. getHotImageList 1246
23.20. getImageList 1246
23.21. getPressedImageList 1247
23.22. indexTocommand 1247
23.23. insertButton 1248
23.24. isButtonChecked 1248
23.25. isButtonEnabled 1249
23.26. isButtonHidden 1249
23.27. isButtonHighlighted 1250
23.28. isButtonIndeterminate 1250
23.29. isButtonPressed 1251
23.30. loadImages 1252
23.31. markButton 1252
23.32. setBitmapSize 1253
23.33. setButtonText 1253
23.34. setDisabledImageList 1254
23.35. setExtendedStyle 1254
23.36. setHotImageList 1255
23.37. setImageList 1255
23.38. setPressedImageList 1256
23.39. TbButton Class 1256
23.39.1. Method Table 1257
23.39.2. new (Class Method) 1257

23.39.3. bitmapID (Attribute) ... 1258
23.39.4. cmdID (Attribute) ... 1258
23.39.5. itemData (Attribute) ... 1259
23.39.6. state (Attribute) ... 1259
23.39.7. style (Attribute) ... 1259
23.39.8. text (Attribute) ... 1260
23.39.9. assignBitmapID ... 1260

24. ToolTip Controls **1262**
24.1. Tool Identification ... 1262
24.2. Method Table ... 1263
24.3. connectToolTipEvent (dialog object method) ... 1265
24.4. createToolTip (dialog object method) ... 1265
24.5. newToolTip (dialog object method) ... 1265
24.6. activate ... 1265
24.7. addTool ... 1266
24.8. addToolEx ... 1268
24.9. addToolRect ... 1268
24.10. adjustRect ... 1270
24.11. delTool ... 1271
24.12. enumTools ... 1272
24.13. getBubbleSize ... 1273
24.14. getCurrentToolInfo ... 1273
24.15. getDelayTime ... 1274
24.16. getMargin ... 1274
24.17. getMaxTipWidth ... 1275
24.18. getText ... 1275
24.19. getTipBkColor ... 1276
24.20. getTipTextColor ... 1276
24.21. getTitle ... 1277
24.22. getToolCount ... 1278
24.23. getToolInfo ... 1278
24.24. hasCurrentTool ... 1279
24.25. hitTestInfo ... 1279
24.26. manageAtypicalTool ... 1281
24.27. newToolRect ... 1283
24.28. pop ... 1284
24.29. popUp ... 1284
24.30. setDelayTime ... 1284
24.31. setMargin ... 1285
24.32. setMaxTipWidth ... 1286
24.33. setTipBkColor ... 1287
24.34. setTipTextColor ... 1287
24.35. setTitle ... 1288
24.36. setToolInfo ... 1289
24.37. setWindowTheme ... 1290
24.38. trackActivate ... 1290
24.39. trackPosition ... 1291
24.40. update ... 1292
24.41. updateTipText ... 1292
24.42. ToolInfo Class ... 1292
24.42.1. Method Table ... 1293
24.42.2. forHitTest (Class Method) ... 1293
24.42.3. forID (Class Method) ... 1294

24.42.4. new (Class Method) ... 1295
24.42.5. flags (Attribute) ... 1297
24.42.6. rect (Attribute) ... 1298
24.42.7. resource (Attribute) ... 1299
24.42.8. rexxHwnd (Attribute) ... 1299
24.42.9. rexxID (Attribute) ... 1300
24.42.10. text (Attribute) ... 1300
24.42.11. userData (Attribute) ... 1300

25. Trackbar Controls 1302

25.1. Method Table ... 1302
25.2. TrackBar Constants ... 1303
25.3. newTrackBar (dialog object method) ... 1304
25.4. createTrackBar (UserDialog method) ... 1304
25.5. connectTrackBarEvent (dialog object method) ... 1304
25.6. clearSelRange ... 1304
25.7. clearTicks ... 1305
25.8. countTicks ... 1305
25.9. getLineStep ... 1305
25.10. getPageStep ... 1305
25.11. getTick ... 1306
25.12. initRange ... 1306
25.13. initSelRange ... 1307
25.14. pos ... 1307
25.15. pos= ... 1308
25.16. range ... 1308
25.17. selRange ... 1308
25.18. setLineStep ... 1308
25.19. setMax ... 1309
25.20. setMin ... 1309
25.21. setPageStep ... 1310
25.22. setPos ... 1310
25.23. setSelEnd ... 1311
25.24. setSelStart ... 1311
25.25. setTickAt ... 1312
25.26. setTickFrequency ... 1312

26. Tree View Controls 1314

26.1. Method Table ... 1314
26.2. newTreeView (dialog object method) ... 1316
26.3. createTreeView (UserDialog method) ... 1316
26.4. connectTreeViewEvent (dialog object method) ... 1316
26.5. add ... 1316
26.6. Child ... 1318
26.7. Collapse ... 1319
26.8. CollapseAndReset ... 1319
26.9. delete ... 1320
26.10. deleteAll ... 1320
26.11. DropHighlight ... 1321
26.12. dropHighlighted ... 1321
26.13. edit ... 1321
26.14. endEdit ... 1322
26.15. ensureVisible ... 1322
26.16. Expand ... 1322
26.17. find ... 1323

26.18. firstVisible 1324
26.19. getImageList 1324
26.20. getItemData 1325
26.21. getItemHeight 1326
26.22. getItemRect 1326
26.23. getStateImage 1327
26.24. getToolTips 1327
26.25. hitTestInfo 1328
26.26. Indent 1329
26.27. Indent= 1329
26.28. insert 1330
26.29. IsAncestor 1332
26.30. itemInfo 1332
26.31. items 1333
26.32. itemText 1334
26.33. MakeFirstVisible 1334
26.34. modify 1335
26.35. moveItem 1337
26.36. next 1337
26.37. NextVisible 1338
26.38. Parent 1338
26.39. previous 1338
26.40. PreviousVisible 1339
26.41. removeItemData 1339
26.42. root 1340
26.43. select 1340
26.44. selected 1340
26.45. setImageList 1341
26.46. setItemData 1342
26.47. setItemHeight 1343
26.48. setStateImage 1343
26.49. setToolTips 1344
26.50. SortChildren 1344
26.51. sortChildrenCB 1345
26.52. Toggle 1347
26.53. VisibleItems 1347
26.54. TvCustomDrawSimple Class 1348
26.54.1. CustomDraw Event Handler 1348
26.54.2. Method Table 1349
26.54.3. new (Class Method) 1350
26.54.4. clrText (Attribute) 1350
26.54.5. clrTextBk (Attribute) 1350
26.54.6. drawStage (Attribute) 1351
26.54.7. font (Attribute) 1351
26.54.8. id (Attribute) 1352
26.54.9. item (Attribute) 1353
26.54.10. itemData (Attribute) 1353
26.54.11. level (Attribute) 1354
26.54.12. reply (Attribute) 1355

27. Up Down Controls 1356
27.1. Method Table 1356
27.2. newUpDown (dialog object method) 1357
27.3. createUpDown (UserDialog method) 1357

27.4. connectUpDownEvent (dialog object method) 1357
27.5. deltaPosReply (Class method) 1357
27.6. getAcceleration 1358
27.7. getBase 1359
27.8. getBuddy 1359
27.9. getPosition 1360
27.10. getRange 1360
27.11. setAcceleration 1361
27.12. setBase 1362
27.13. setBuddy 1362
27.14. setPosition 1363
27.15. setRange 1364

28. Menus 1366

28.1. Menu Item IDs 1367
28.2. Menu Command Items 1367
28.2.1. Menu Command Events 1367
28.2.2. Menu Command Event Connections 1368
28.2.3. Menu Command Event Handlers 1371
28.3. The Menu Object 1374
28.3.1. Method Table 1374
28.3.2. connectCommandEvent (Class method) 1376
28.3.3. hMenu (Attribute) 1377
28.3.4. wID (Attribute) 1378
28.3.5. check 1378
28.3.6. checkRadio 1379
28.3.7. connectAllCommandEvents 1380
28.3.8. connectCommandEvent 1381
28.3.9. connectMenuEvent 1382
28.3.10. connectSomeCommandEvents 1388
28.3.11. deletePopup 1389
28.3.12. destroy 1390
28.3.13. disable 1391
28.3.14. enable 1391
28.3.15. getAutoConnectStatus 1392
28.3.16. getCount 1393
28.3.17. getHandle 1393
28.3.18. getHelpID 1394
28.3.19. getID 1394
28.3.20. getItemState 1395
28.3.21. getMaxHeight 1396
28.3.22. getMenuHandle 1397
28.3.23. getPopup 1397
28.3.24. getText 1398
28.3.25. getType 1399
28.3.26. gray 1400
28.3.27. hilite 1401
28.3.28. insertItem 1402
28.3.29. insertPopup 1404
28.3.30. insertSeparator 1406
28.3.31. isChecked 1407
28.3.32. isCommandItem 1408
28.3.33. isDisabled 1408
28.3.34. isEnabled 1409

28.3.35. isGrayed 1410
28.3.36. isPopup 1410
28.3.37. isSeparator 1411
28.3.38. isValidItemID 1411
28.3.39. isValidMenu 1413
28.3.40. isValidMenuHandle 1413
28.3.41. itemTextToMethodName 1413
28.3.42. releaseMenuHandle 1414
28.3.43. removeItem 1415
28.3.44. removePopup 1415
28.3.45. removeSeparator 1416
28.3.46. setAutoConnection 1417
28.3.47. setHelpID 1418
28.3.48. setID 1419
28.3.49. setMaxHeight 1420
28.3.50. setText 1421
28.3.51. uncheck 1421
28.3.52. unhilite 1422
28.4. The Menu Bar Object 1423
28.4.1. Method Table 1423
28.4.2. attachTo 1423
28.4.3. detach 1425
28.4.4. isAttached 1425
28.4.5. redraw 1425
28.4.6. replace 1426
28.5. BinaryMenuBar Class 1427
28.5.1. Method Table 1427
28.5.2. new (Class) 1427
28.6. ScriptMenuBar Class 1429
28.6.1. Method Table 1429
28.6.2. new (Class) 1429
28.7. UserMenuBar Class 1431
28.7.1. Method Table 1431
28.7.2. new (Class) 1432
28.7.3. addItem 1433
28.7.4. addPopup 1435
28.7.5. addSeparator 1437
28.7.6. complete 1438
28.7.7. isComplete 1439
28.8. PopupMenu Class 1439
28.8.1. ContextMenu Event Handler 1439
28.8.2. Method Table 1441
28.8.3. connectContextMenu (Class) 1441
28.8.4. new (Class) 1442
28.8.5. assignTo 1443
28.8.6. connectContextMenu 1444
28.8.7. isAssigned 1446
28.8.8. show 1446
28.8.9. track 1449
28.9. SystemMenu Class 1452
28.9.1. Method Table 1452
28.9.2. Constant Methods 1453
28.9.3. new (Class) 1454
28.9.4. connectAllCommandEvents 1454

28.9.5. connectCommandEvent ... 1455
28.9.6. revert ... 1456
28.9.7. connectSomeCommandEvents ... 1457

29. Resources 1459

29.1. Resource Classes ... 1459
29.2. Image Class ... 1459
29.2.1. Method Table ... 1460
29.2.2. colorRef (Class Method) ... 1461
29.2.3. fromFiles (Class method) ... 1461
29.2.4. fromIDs (Class method) ... 1463
29.2.5. getBValue (Class Method) ... 1464
29.2.6. getGValue (Class Method) ... 1464
29.2.7. getImage (Class method) ... 1464
29.2.8. getRValue (Class Method) ... 1466
29.2.9. new (Class method) ... 1467
29.2.10. toID (Class method) ... 1467
29.2.11. userIcon (Class method) ... 1468
29.2.12. handle ... 1469
29.2.13. release ... 1470
29.2.14. isNull ... 1471
29.2.15. systemErrorCode ... 1472
29.3. ImageList Class ... 1473
29.3.1. Method Table ... 1473
29.3.2. create (Class method) ... 1474
29.3.3. new (Class method) ... 1476
29.3.4. add ... 1476
29.3.5. addIcon ... 1477
29.3.6. addImages ... 1477
29.3.7. addMasked ... 1479
29.3.8. duplicate ... 1480
29.3.9. getCount ... 1481
29.3.10. getIcon ... 1481
29.3.11. getImageSize ... 1483
29.3.12. handle ... 1484
29.3.13. isNull ... 1484
29.3.14. release ... 1485
29.3.15. remove ... 1485
29.3.16. removeAll ... 1486
29.4. ResourceImage Class ... 1487
29.4.1. Method Table ... 1488
29.4.2. new (Class method) ... 1488
29.4.3. getImage ... 1489
29.4.4. getImages ... 1491
29.4.5. handle ... 1493
29.4.6. isNull ... 1494
29.4.7. release ... 1494
29.4.8. systemErrorCode ... 1495

30. User Input 1497

30.1. Common Dialog Library ... 1497
30.1.1. Class and Routine Table ... 1497
30.1.2. Standard Dialog Classes ... 1498
30.1.3. Public Routines ... 1528
30.2. Mouse Class ... 1541

30.2.1. Hit Test Keywords 1541
30.2.2. Method Table 1542
30.2.3. Constant Methods 1543
30.2.4. new (Class Method) 1544
30.2.5. doubleClickTime (Class Method) 1545
30.2.6. loadCursor (Class Method) 1545
30.2.7. loadCursorFromFile (Class Method) 1547
30.2.8. setDoubleClickTime (Class Method) 1548
30.2.9. swapButton (Class Method) 1548
30.2.10. appStarting 1549
30.2.11. arrow 1551
30.2.12. capture 1551
30.2.13. clipCursor 1552
30.2.14. connectEvent 1553
30.2.15. cross 1564
30.2.16. dragDetect 1564
30.2.17. getCapture 1565
30.2.18. getClipCursor 1566
30.2.19. getCursorPos 1566
30.2.20. isButtonDown 1567
30.2.21. no 1568
30.2.22. releaseCapture 1568
30.2.23. releaseClipCursor 1569
30.2.24. restoreCursor 1570
30.2.25. setCursor 1571
30.2.26. setCursorPos 1572
30.2.27. showCursor 1573
30.2.28. trackEvent 1574
30.2.29. wait 1577
30.3. Keyboard Class 1577
30.3.1. Method Table 1578
30.3.2. getAsyncKeyState (Class Method) 1578

31. Utility Classes, Routines, and Objects 1580
31.1. .application object 1581
31.2. ApplicationManager Class 1581
31.2.1. Method Table 1581
31.2.2. new 1582
31.2.3. constDir (Attribute) 1582
31.2.4. srcDir (Attribute) 1582
31.2.5. addToConstDir 1584
31.2.6. autoDetection 1585
31.2.7. defaultFont 1585
31.2.8. defaultIcon 1586
31.2.9. parseIncludeFile 1587
31.2.10. requiredOS 1587
31.2.11. resolveNumericID 1588
31.2.12. resolveSymbolicID 1588
31.2.13. setDefaults 1588
31.2.14. useGlobalConstDir 1590
31.3. .constDir object 1592
31.4. DlgUtil Class 1593
31.4.1. Method Table 1593
31.4.2. and (Class Method) 1594

31.4.3. comCtl32Version (Class Method) 1595
31.4.4. errMsg 1596
31.4.5. getGUID 1597
31.4.6. getSystemMetrics (Class Method) 1598
31.4.7. hiWord (Class Method) 1599
31.4.8. loWord (Class Method) 1599
31.4.9. makeLParam (Class Method) 1600
31.4.10. makeWParam (Class Method) 1601
31.4.11. or (Class Method) 1602
31.4.12. screenArea (Class Method) 1603
31.4.13. screenSize (Class Method) 1603
31.4.14. shiftLeft 1604
31.4.15. shiftRight 1604
31.4.16. sHiWord (Class Method) 1605
31.4.17. signed 1605
31.4.18. signed32 1606
31.4.19. sLoWord (Class Method) 1606
31.4.20. sShiftLeft 1607
31.4.21. sShiftRight 1608
31.4.22. threadID (Class Method) 1608
31.4.23. unsigned 1609
31.4.24. unsigned32 1610
31.4.25. version (Class Method) 1611
31.4.26. windowFromPoint 1612
31.5. FindWindow Routine 1612
31.6. Locate 1613
31.7. MSSleep Routine 1614
31.8. OS Class 1614
31.8.1. Method Table 1614
31.8.2. is32on64bit (Class Method) 1615
31.8.3. is64bit (Class Method) 1615
31.8.4. isAtLeastVista (Class Method) 1615
31.8.5. isAtLeastW2K (Class Method) 1616
31.8.6. isAtLeastW2K3 (Class Method) 1616
31.8.7. isAtLeastWindows7 (Class Method) 1616
31.8.8. isAtLeastXP (Class Method) 1616
31.8.9. isServer2008 (Class Method) 1616
31.8.10. isServer2008R2 (Class Method) 1617
31.8.11. isVista (Class Method) 1617
31.8.12. isW2K (Class Method) 1617
31.8.13. isW2K3 (Class Method) 1617
31.8.14. isWindows7 (Class Method) 1617
31.8.15. isWow64 (Class Method) 1617
31.8.16. isXP (Class Method) 1618
31.8.17. isXP32 (Class Method) 1618
31.8.18. isXP64 (Class Method) 1618
31.8.19. settingChanged 1618
31.8.20. shellChangeNotify 1619
31.9. Play Routine 1621
31.10. Point Class 1621
31.10.1. Method Table 1622
31.10.2. new (Class Method) 1622
31.10.3. x (Attribute) 1623
31.10.4. y (Attribute) 1623

31.10.5. + (addition) 1624
31.10.6. - (subtraction) 1624
31.10.7. copy 1625
31.10.8. decr 1626
31.10.9. incr 1626
31.10.10. inRect 1627
31.10.11. string 1628
31.11. Rect Class 1628
31.11.1. Method Table 1628
31.11.2. new (Class Method) 1629
31.11.3. bottom (Attribute) 1630
31.11.4. left (Attribute) 1630
31.11.5. right (Attribute) 1631
31.11.6. top (Attribute) 1631
31.11.7. copy 1631
31.11.8. string 1632
31.12. ScreenSize Routine 1633
31.13. Size Class 1633
31.13.1. Method Table 1633
31.13.2. new (Class Method) 1634
31.13.3. height (Attribute) 1635
31.13.4. width (Attribute) 1635
31.13.5. Comparison Operators 1636
31.13.6. equateTo 1637
31.13.7. string 1637
31.14. SM Class 1638
31.14.1. Method Table 1638
31.14.2. cMouseButtons (Attribute) 1639
31.14.3. cxCursor (Attribute) 1639
31.14.4. cxDrag (Attribute) 1639
31.14.5. cxFixedFrame (Attribute) 1640
31.14.6. cxIcon (Attribute) 1640
31.14.7. cxScreen (Attribute) 1640
31.14.8. cxSize (Attribute) 1641
31.14.9. cxSmIcon (Attribute) 1641
31.14.10. cxVscroll (Attribute) 1641
31.14.11. cyCaption (Attribute) 1641
31.14.12. cyCursor (Attribute) 1642
31.14.13. cyDrag (Attribute) 1642
31.14.14. cyFixedFrame (Attribute) 1642
31.14.15. cyHscroll (Attribute) 1643
31.14.16. cyIcon (Attribute) 1643
31.14.17. cyMenu (Attribute) 1643
31.14.18. cyScreen (Attribute) 1643
31.14.19. cySize (Attribute) 1644
31.14.20. cySmIcon (Attribute) 1644
31.14.21. menuDropAlignment (Attribute) 1644
31.15. SPI Class 1644
31.15.1. Method Table 1645
31.15.2. WHEEL_PAGESCROLL (Constant) 1645
31.15.3. dragHeight (Attribute) 1645
31.15.4. dragWidth (Attribute) 1646
31.15.5. menuAnimation (Attribute) 1646
31.15.6. menuFade (Attribute) 1647

31.15.7. mouseHoverHeight (Attribute) 1647
31.15.8. mouseHoverTime (Attribute) 1648
31.15.9. mouseHoverWidth (Attribute) 1648
31.15.10. updateFlag (Attribute) 1649
31.15.11. wheelScrollLines (Attribute) 1650
31.15.12. workArea (Attribute) 1650
31.16. .systemErrorCode object 1651
31.17. VK Class 1652
31.17.1. Method Table 1653
31.17.2. key2name 1658
31.17.3. Example VK Constant Use 1659
31.18. Window Class 1659
31.18.1. Method Table 1660
31.18.2. new (Class Method) 1661
31.18.3. initCode (Attribute) 1662
31.18.4. hwnd (Attribute) 1662
31.18.5. factorX (Attribute) 1662
31.18.6. factorY (Attribute) 1662
31.18.7. pixelCX (Attribute) 1662
31.18.8. pixelCY (Attribute) 1663
31.18.9. sizeX (Attribute) 1663
31.18.10. sizeY (Attribute) 1663
31.18.11. clear 1663
31.18.12. client2screen 1663
31.18.13. clientRect 1663
31.18.14. clientToScreen 1664
31.18.15. disable 1664
31.18.16. display 1664
31.18.17. draw 1664
31.18.18. enable 1664
31.18.19. foregroundWindow 1664
31.18.20. getClientRect 1664
31.18.21. getExStyleRaw 1665
31.18.22. getID 1665
31.18.23. getPos 1665
31.18.24. getRealPos 1665
31.18.25. getRealSize 1665
31.18.26. getRect 1665
31.18.27. getSize 1665
31.18.28. getStyleRaw 1666
31.18.29. getText 1666
31.18.30. getTextSizePx 1666
31.18.31. getTextSizeScreen 1666
31.18.32. hide 1666
31.18.33. hideFast 1666
31.18.34. isEnabled 1666
31.18.35. isVisible 1667
31.18.36. move 1667
31.18.37. moveTo 1667
31.18.38. redraw 1667
31.18.39. redrawClient 1667
31.18.40. resize 1667
31.18.41. resizeTo 1668
31.18.42. screen2client 1668

31.18.43. screenToClient 1668
31.18.44. sendMessage 1668
31.18.45. sendMessageHandle 1668
31.18.46. setRect 1668
31.18.47. setStyleRaw 1669
31.18.48. setText 1669
31.18.49. setTitle 1669
31.18.50. showFast 1669
31.18.51. title 1669
31.18.52. title= 1670
31.18.53. update 1670
31.18.54. windowRect 1670
31.19. WinTimer Routine 1670

32. Windows Shell Objects 1672
32.1. Shell Object Classes Table 1672
32.2. Shell / COM considerations 1672
32.3. Special Folder Names 1675
32.4. BrowseForFolder Class 1679
32.4.1. Method Table 1679
32.4.2. new (Class Method) 1680
32.4.3. banner (Attribute) 1681
32.4.4. dlgTitle (Attribute) 1681
32.4.5. hint (Attribute) 1682
32.4.6. initialThread (Attribute) 1683
32.4.7. options (Attribute) 1685
32.4.8. owner (Attribute) 1687
32.4.9. root (Attribute) 1688
32.4.10. startDir (Attribute) 1689
32.4.11. usePathForHint (Attribute) 1689
32.4.12. getDisplayName 1690
32.4.13. getFolder 1691
32.4.14. getItemIDList 1692
32.4.15. initCOM 1694
32.4.16. releaseCOM 1695
32.4.17. releaseItemIDList 1697
32.5. ComConstants Mixin Class 1699
32.5.1. Constant Method Table 1699
32.6. CommonDialogCustomizations Class 1699
32.6.1. Method Table 1700
32.6.2. addCheckButton 1701
32.6.3. addComboBox 1702
32.6.4. addControlItem 1702
32.6.5. addEditBox 1703
32.6.6. addMenu 1704
32.6.7. addPushButton 1705
32.6.8. addRadioButtonList 1706
32.6.9. addSeparator 1706
32.6.10. addText 1707
32.6.11. check 1708
32.6.12. enableOpenDropDown 1708
32.6.13. endVisualGroup 1709
32.6.14. getCheckButtonState 1710
32.6.15. getControlItemState 1710

32.6.16. getControlState 1711
32.6.17. getEditBoxText 1712
32.6.18. getSelectedControlItem 1713
32.6.19. isChecked 1714
32.6.20. makeProminent 1714
32.6.21. removeAllControlItems 1715
32.6.22. removeControlItem 1716
32.6.23. setCheckButtonState 1716
32.6.24. setControlItemState 1717
32.6.25. setControlItemText 1718
32.6.26. setControlLabel 1718
32.6.27. setControlState 1719
32.6.28. setEditBoxText 1720
32.6.29. setSelectedControlItem 1721
32.6.30. startVisualGroup 1721
32.6.31. uncheck 1722
32.7. CommonItemDialog Class 1723
32.7.1. Method Table 1723
32.7.2. new (Class Method) 1724
32.7.3. options (Attribute) 1725
32.7.4. addPlace 1727
32.7.5. advise 1728
32.7.6. clearClientData 1729
32.7.7. close 1729
32.7.8. getCurrentSelection 1731
32.7.9. getFileName 1732
32.7.10. getFileTypeIndex 1733
32.7.11. getFolder 1733
32.7.12. getResult 1734
32.7.13. initCOM 1735
32.7.14. isReleased 1737
32.7.15. release 1737
32.7.16. releaseCOM 1739
32.7.17. setCancelButtonLabel 1740
32.7.18. setClientGuid 1741
32.7.19. setDefaultExtension 1742
32.7.20. setDefaultFolder 1743
32.7.21. setFileName 1744
32.7.22. setFileNameLabel 1744
32.7.23. setFileTypeIndex 1745
32.7.24. setFileTypes 1746
32.7.25. setFilter 1747
32.7.26. setFolder 1747
32.7.27. setOkButtonLabel 1748
32.7.28. setTitle 1749
32.7.29. show 1750
32.7.30. unadvise 1750
32.8. CommonDialogEvents Class 1751
32.8.1. Method Table 1751
32.8.2. Constant Methods 1752
32.8.3. new (Class Method) 1752
32.8.4. onButtonClicked 1753
32.8.5. onCheckButtonToggled 1754
32.8.6. onControlActivating 1754

32.8.7. onFileOk 1755
32.8.8. onFolderChange 1756
32.8.9. onFolderChanging 1757
32.8.10. onHelp 1758
32.8.11. onItemSelected 1758
32.8.12. onOverwrite 1760
32.8.13. onSelectionChange 1761
32.8.14. onShareViolation 1761
32.8.15. onTypeChange 1762
32.9. OpenFileDialog Class 1763
32.9.1. Method Table 1764
32.9.2. new (Class Method) 1765
32.9.3. getResults 1765
32.10. SaveFileDialog Class 1766
32.10.1. Method Table 1767
32.10.2. new (Class Method) 1768
32.10.3. setSaveAsItem 1768
32.11. ShellItemFilter Class 1769
32.11.1. Method Table 1769
32.11.2. new (Class Method) 1770
32.11.3. includeItem 1770
32.12. SimpleFolderBrowse Class 1771
32.12.1. Method Table 1771
32.12.2. getFolder (Class Method) 1772
32.13. SimpleFolderBrowse Routine 1773

A. Deprecated Classes, Methods, Functions, and Routines 1775

A.1. Deprecated Classes 1775
A.1.1. AdvancedControls Mixin Class 1775
A.1.2. MessageExtensions Mixin Class 1776
A.1.3. PlainUserDialog Class 1776
A.1.4. CategoryDialog Class 1776
A.1.5. PropertySheet Class 1776
A.2. Deprecated Dialog Methods 1776
A.2.1. captureMouse 1776
A.2.2. checkMenuItem 1776
A.2.3. clearButtonRect 1776
A.2.4. clearMessages 1777
A.2.5. combineELwithSB 1777
A.2.6. connectAnimatedButton 1777
A.2.7. connectBitmapButton 1777
A.2.8. connectEntryLine 1777
A.2.9. connectListControl 1777
A.2.10. connectMultiListBox 1777
A.2.11. connectSliderControl 1777
A.2.12. connectTabControl 1777
A.2.13. connectTreeControl 1777
A.2.14. deInstall 1777
A.2.15. disableItem 1777
A.2.16. disableMenuItem 1777
A.2.17. dump 1778
A.2.18. enableItem 1778
A.2.19. enableMenuItem 1778
A.2.20. freeButtonDC 1778

A.2.21. focusItem 1778
A.2.22. getButtonControl 1778
A.2.23. getButtonDC 1778
A.2.24. getButtonRect 1778
A.2.25. getCheckBox 1778
A.2.26. getCheckControl 1778
A.2.27. getComboBox 1778
A.2.28. getComboLine 1778
A.2.29. getEditControl 1778
A.2.30. getEntryLine 1779
A.2.31. getGroupBox 1779
A.2.32. getItem 1779
A.2.33. getListBox 1779
A.2.34. getListControl 1779
A.2.35. getListLine 1779
A.2.36. getMenuItemState 1779
A.2.37. getMouseCapture 1779
A.2.38. getMultiList 1779
A.2.39. getProgressBar 1779
A.2.40. getRadioButton 1779
A.2.41. getRadioControl 1779
A.2.42. getScrollBar 1779
A.2.43. getSliderControl 1779
A.2.44. getStaticControl 1780
A.2.45. getTabControl 1780
A.2.46. getTextSize 1780
A.2.47. getTreeControl 1780
A.2.48. getValue 1780
A.2.49. getWindowRect 1780
A.2.50. grayMenuItem 1780
A.2.51. handleMessages 1780
A.2.52. hideItem 1780
A.2.53. hideItemFast 1780
A.2.54. isMouseButtonDown 1780
A.2.55. itemText 1780
A.2.56. moveItem 1780
A.2.57. peekDialogMessage 1781
A.2.58. redrawItem 1781
A.2.59. releaseMouseCapture 1781
A.2.60. resizeItem 1781
A.2.61. run 1781
A.2.62. scrollInButton 1781
A.2.63. setItemFont 1781
A.2.64. setItemColor 1781
A.2.65. setItemSysColor 1781
A.2.66. setMenuItemRadio 1781
A.2.67. setStaticText 1781
A.2.68. uncheckMenuItem 1781
A.2.69. writeToButton 1782
A.3. Deprecated Dialog Control Methods 1782
A.3.1. Deprecated Button Methods 1782
A.3.2. Deprecated ComboBox Methods 1783
A.3.3. Deprecated ListView Methods 1783
A.3.4. Deprecated ProgressBar Methods 1784

A.3.5. Deprecated Radio Button Methods 1784
A.3.6. Deprecated Tab Methods 1784
A.3.7. Deprecated TreeView Methods 1784
A.4. Deprecated DialogControl Methods 1785
A.4.1. captureMouse 1785
A.4.2. getFocus 1785
A.4.3. getMouseCapture 1785
A.4.4. getTextSize 1785
A.4.5. isMouseButtonDown 1786
A.4.6. processMessage 1786
A.4.7. releaseMouseCapture 1786
A.4.8. setFocus 1786
A.4.9. tabToNext 1786
A.4.10. tabToPrevious 1786
A.4.11. value 1786
A.4.12. value= 1786
A.5. Deprecated Event Notification Methods 1786
A.5.1. connectButton 1786
A.5.2. connectButtonNotify 1787
A.5.3. connectComboBoxNotify 1787
A.5.4. connectCommonNotify 1787
A.5.5. connectControl 1787
A.5.6. connectEditNotify 1787
A.5.7. connectList 1787
A.5.8. connectListBoxNotify 1787
A.5.9. connectListLeftDoubleClick 1787
A.5.10. connectListNotify 1787
A.5.11. connectListViewNotify 1787
A.5.12. connectMenuItem 1788
A.5.13. connectMouseCapture 1788
A.5.14. connectScrollBar 1788
A.5.15. connectScrollBarNotify 1788
A.5.16. connectSliderNotify 1788
A.5.17. connectStaticNotify 1788
A.5.18. connectTabNotify 1788
A.5.19. connectTreeNotify 1788
A.6. Deprecated External Functions 1788
A.6.1. errorMessage 1789
A.6.2. getFileNameWindow 1789
A.6.3. getScreenSize 1789
A.6.4. getSysMetrics 1789
A.6.5. infoMessage 1789
A.6.6. playSoundFile 1789
A.6.7. playSoundFileInLoop 1789
A.6.8. sleepMS 1790
A.6.9. stopSoundFile 1790
A.6.10. systemMetrics Routine 1790
A.6.11. yesNoMessage 1790
A.7. Deprecated UserDialog Methods 1790
A.7.1. addBitmapButton 1790
A.7.2. addBlackFrame 1790
A.7.3. addBlackRect 1790
A.7.4. addButton 1790
A.7.5. addButtonGroup 1790

A.7.6. addCheckBox 1790
A.7.7. addCheckBoxStem 1790
A.7.8. addCheckGroup 1791
A.7.9. addComboBox 1791
A.7.10. addComboInput 1791
A.7.11. addEntryLine 1791
A.7.12. addEtchedFrame 1791
A.7.13. addEtchedHorizontal 1791
A.7.14. addEtchedVertical 1791
A.7.15. addGrayFrame 1791
A.7.16. addGrayRect 1791
A.7.17. addGroupBox 1791
A.7.18. addIcon 1791
A.7.19. addImage 1791
A.7.20. addInput 1791
A.7.21. addInputGroup 1791
A.7.22. addInputStem 1792
A.7.23. addListBox 1792
A.7.24. addListControl 1792
A.7.25. addMenuItem 1792
A.7.26. addMenuSeparator 1792
A.7.27. addOkCancelLeftBottom 1792
A.7.28. addOkCancelLeftTop 1792
A.7.29. addOkCancelRightBottom 1792
A.7.30. addOkCancelRightTop 1792
A.7.31. addPasswordLine 1792
A.7.32. addPopupMenu 1792
A.7.33. addProgressBar 1792
A.7.34. addRadioButton 1792
A.7.35. addRadioGroup 1792
A.7.36. addRadioStem 1793
A.7.37. addScrollBar 1793
A.7.38. addSliderControl 1793
A.7.39. addStatic 1793
A.7.40. addTabControl 1793
A.7.41. addText 1793
A.7.42. addTreeControl 1793
A.7.43. addWhiteFrame 1793
A.7.44. addWhiteRect 1793
A.7.45. createMenu 1793
A.7.46. loadMenu 1793
A.7.47. setMenu 1793
A.8. Deprecated WindowBase Methods 1793
A.8.1. assignWindow 1794
A.8.2. getSystemMetrics 1794
A.9. Deprecated WindowExtensions Methods 1794
A.9.1. absRect2LogRect 1794
A.9.2. cursor_appStarting 1794
A.9.3. cursor_arrow 1794
A.9.4. cursor_cross 1794
A.9.5. cursor_no 1794
A.9.6. cursor_wait 1794
A.9.7. cursorPos 1794
A.9.8. logRect2AbsRect 1795

A.9.9. restoreCursorShape 1795
A.9.10. setCursorPos 1795

B. Notices **1796**

B.1. Trademarks 1796
B.2. Source Code For This Document 1797

C. Common Public License Version 1.0 **1798**

C.1. Definitions 1798
C.2. Grant of Rights 1798
C.3. Requirements 1799
C.4. Commercial Distribution 1799
C.5. No Warranty 1800
C.6. Disclaimer of Liability 1800
C.7. General 1800

D. Revision History **1802**

Index **1803**

Preface

This book describes the ooDialog framework, which is implemented as an external library package, and is part of the Open Object Rexx distribution on the Windows® platform. It describes the classes in the framework and how to use the framework to program graphical user interfaces, (commonly referred to as a GUI,) on Windows

This book is intended for Open Object Rexx programmers who want to design graphical user interfaces for their applications.

1. How This Book is Structured

This book is primarily a reference that describes the classes and methods in the ooDialog framework in detail. In general, each chapter describes a single class and its methods. Although, in some cases similar classes that do not need a lot of documentation are gathered together in a single chapter with a subsection for each class. For the most part, the documentation for each class will have a table, at the beginning, that lists the methods of the class, with a link to the detailed documentation for the method within that chapter or section.

One slight deviation from that pattern is found in the *dialog object* and *dialog control object* chapters at the beginning of the book. These objects are composed of a number of base classes and mixin classes. These two chapters list the class methods, attributes, and instance methods that are a part of every dialog, or every dialog control object, respectively, without much distinction as to exactly which base, or mixin, class the method comes from. The method tables at the beginning of the chapters will contain links to the detailed documentation, which may be in another chapter.

The *Preface* and Chapter 1, contain general information that should be read, or re-read, with every new release. The rest of the book is wholly reference and the reader can navigate to the specific topic they are interested in. The *Preface*, (this section) is intended to give the reader a better understanding of how to use this reference. As the book is revised, how best to use the book may change. Chapter 1, a *Brief Overview*, contains general information that all users of ooDialog should know. This content is dynamic. In particular there is a section on the *current release* that will contain important information the user should be aware of in the new release. The chapter also contains a section on *common concepts* that all ooDialog programmers should be aware of. As new features are added, or new misconceptions come to light, this information is also likely to change over time.

2. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

2.1. Typographic Conventions

Typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold` is used to highlight literal strings, class names, or inline code examples. For example:

> The **`Class`** class comparison methods return **`.true`** or **`.false`**, the result of performing the comparison operation.
>
> This method is exactly equivalent to **`subWord(`***`n`***`, 1)`**.

`Mono-spaced Normal` denotes method names or source code in program listings set off as separate examples.

This method has no effect on the action of any `hasEntry`, `hasIndex`, `items`, `remove`, or `supplier` message sent to the collection.

```
-- reverse an array
a = .Array~of("one", "two", "three", "four", "five")

-- five, four, three, two, one
aReverse = .CircularQueue~new(a~size)~appendAll(a)~makeArray("lifo")
```

Proportional Italic is used for method and function variables and arguments.

A supplier loop specifies one or two control variables, *index*, and *item*, which receive a different value on each repetition of the loop.

Returns a string of length *length* with *string* centered in it and with *pad* characters added as necessary to make up length.

2.2. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed, like mandatory initialization. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.

Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

3. How to Read the Syntax Diagrams

Throughout this book, syntax is described using the structure defined below. This is similar to, but slightly different than, the IBM syntax diagrams used in other ooRexx reference documentation. The author is calling these diagrams *simplified railroad tracks*. It primarily strives to limit all diagrams to 2 lines, and does away with much of the complexity of true IBM railroad tracks. The body of text following the syntax diagrams, along with this clarifying text, will resolve any ambiguities in the diagrams.

Note: Not all syntax diagrams may be converted to the simplified railroad tracks at this time.

In this reference, the syntax diagrams being presented are the diagrams of the syntax for method, attribute, and routine invocations. The diagrams show the method, attribute, or routine name and the arguments for the invocation. For method or attribute invocations, the section the method or attribute is included in, and / or the text itself, make it clear as to which class the method belongs to. The following defines how the syntax diagrams are to be read:

- Each syntax diagram is for a single method, attribute, or routine invocation,
- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The **`>>--`** symbol indicates the beginning of an invocation or call.

  The **`-->`** symbol indicates that the invocation syntax is continued on the next line. In most cases continuation is avoided.

  The **`>---`** symbol indicates that a statement is continued from the previous line.

  The **`-><`** symbol indicates the end of a statement.
- The method, attribute, or routine name is the first token, (first word,) on the horizontal line (the main path). Parentheses enclose the arguments, if there are any arguments.

```
>>--methodName()---------------------------------><
```

- Methods, attributes, or routines that do not accept arguments, stand alone on the main line.

```
>>--attributeName--------------------------------><
```

- Required arguments appear on the horizontal line, (the main path,) within the parentheses.

```
>>--methodName(--requiredArgument--)-------------><
```

- Optional arguments appear on a line directly below the main path.

```
>>--methodName(--+--------------------+--)-------><
                 +--optionalArgument--+
```

- Commas are placed on the same line and immediately before the argument whose position relies on the comma.

```
>>--methodName(--arg1--+-----------+--,--arg3--+-----------+--)----------------><
                       +--,--arg2--+           +--,--arg4--+
```

Important

In ooRexx, the arguments to methods or functions are enclosed in parenthesis and each argument is separated by a comma. The arguments are *positional* and the commas are always required to properly place an argument in its correct position. Required arguments can not be omitted. Optional arguments can be omitted, however, any argument that follows an omitted argument *must* be in its proper position as delineated by the commas. If there are no arguments following an omitted argument, there is no need to include any following commas. But, it is not incorrect to include the following commas.

In the syntax diagrams in this document, arguments are represented by appropriate variable names and these variable names are then described in the text for the method. When arguments are optional, the default value and or behavior if the argument is omitted is also described in the text.

In **all** cases, the text rather than the syntax diagram should be considered definitive.

The following example shows the described syntax:

```
>>--createFontEx(--fontName--+----------------+--+-----------------+--)--------><
                             +--,--pointSize--+  +--,--additional--+
```

In the above example, the syntax diagram indicates that the name of the method or routine is *createFontEx*. The *location* of the syntax diagram in the body of this document would clearly indicate that it is a method of the *WindowsExtensions* class, that has been inherited by the *dialog* object. The diagram indicates that the argument in the first position, *fontName* is required. The argument in the second postion, *pointSize* is optional, and the argument in the third position, *additional* is also optional.

In addition, the syntax diagram indicates that *if* the second positional argument is omitted but the third positional arugment is used, there **must** be 2 commas preceding the *additional* argument.

Important

Although the reader may prefer to interpret the syntax diagrams in some other way than that just explained, any other interpretation is not correct.

4. Getting Help and Submitting Feedback

The Open Object Rexx Project has a number of methods to obtain help and submit feedback for ooRexx and the extension packages that are part of ooRexx. These methods are listed below. For users of ooDialog, the order listed below is in the order that is most likely to receive prompt help or support.

4.1. The Open Object Rexx SourceForge Site

The *Open Object Rexx Project*[1] utilizes *SourceForge*[2] to house the *ooRexx Project*[3] source repositories, mailing lists, and other project features. Over time it has become apparent to the ooDialog developers that the mailing lists are better tools for carrying on discussions concerning ooDialog and that the Forums provided by SourceForge are cumbersome to use. The ooDialog user is most likely to get timely replies by posting questions or comments to the Users Mailing List.

Here is a list of some of the most useful facilities provided by SourceForge.

The Users Mailing List

: You can subscribe to the oorexx-users mailing list at *ooRexx Mailing List Subscriptions*[4] page. This list is for discussing using ooRexx. It also supports a historical archive of past messages. Users of ooDialog are *most* likely to get a prompt reply to their questions, suggestions, or comments on that mailing list.

The Announcements Mailing List

: You can subscribe to the oorexx-announce mailing list at *ooRexx Mailing List Subscriptions*[5] page. This list is only used to announce significant ooRexx project events.

The Developer Mailing List

: You can subscribe to the oorexx-devel mailing list at *ooRexx Mailing List Subscriptions*[6] page. This list is for discussing ooRexx project development activities and future interpreter enhancements. It also supports a historical archive of past messages.

The Bug Mailing List

: You can subscribe to the oorexx-bugs mailing list at *ooRexx Mailing List Subscriptions*[7] page. This list is only used for monitoring changes to the ooRexx bug tracking system.

Bug Reports

: You can create a bug report at *ooRexx Bug Report*[8] page. Please try to provide as much information in the bug report as possible so that the developers can determine the problem as quickly as possible. Sample programs that can reproduce your problem will make it easier to debug reported problems.

Documentation Feedback

: You can submit feedback for, or report errors in, the documentation at *ooRexx Documentation Report*[9] page. Please try to provide as much information in a documentation report as possible. In addition to listing the document and section the report concerns, direct quotes of the text will help the developers locate the text in the source code for the document. (Section numbers are generated when the document is produced and are not available in the source code itself.) Suggestions as to how to reword or fix the existing text should also be included.

Request For Enhancement

: You can suggest ooRexx features at the *ooRexx Feature Requests*[10] page.

[1] http://www.oorexx.org/
[2] http://sourceforge.net/
[3] http://sourceforge.net/projects/oorexx
[4] http://sourceforge.net/p/oorexx/mailman/
[5] http://sourceforge.net/p/oorexx/mailman/
[6] http://sourceforge.net/p/oorexx/mailman/
[7] http://sourceforge.net/p/oorexx/mailman/
[8] http://sourceforge.net/p/oorexx/bugs/
[9] http://sourceforge.net/p/oorexx/documentation/
[10] http://sourceforge.net/p/oorexx/feature-requests/

Patch Reports

If you create an enhancement patch for ooRexx please post the patch using the *ooRexx Patch Report*[11] page. Please provide as much information in the patch report as possible so that the developers can evaluate the enhancement as quickly as possible.

Please do not post bug fix patches here, instead you should open a bug report and attach the patch to it.

The ooRexx Forums

The ooRexx project maintains a set of forums that anyone may contribute to or monitor. They are located on the *Open Object Rexx Discussion*[12] page. There are currently four forums available: Help, Developers, Open Discussion, and General Discussion. In addition, you can monitor the forums via email.

4.2. The Rexx Language Association Mailing List

The *Rexx Language Association*[13] maintains a mailing list for its members. This mailing list is only available to RexxLA members thus you will need to join RexxLA in order to get on the list. The dues for RexxLA membership are small and are charged on a yearly basis. For details on joining RexxLA please refer to the *RexxLA Home Page*[14] or the *RexxLA Membership Application*[15] page.

4.3. comp.lang.rexx Newsgroup

The *comp.lang.rexx*[16] newsgroup is a good place to obtain help from many individuals within the Rexx community. You can obtain help on Open Object Rexx or on any number of other Rexx interpreters and tools.

5. Related Information

See also: *Open Object Rexx: Reference*

[11] http://sourceforge.net/p/oorexx/patches/
[12] http://sourceforge.net/p/oorexx/discussion/
[13] http://www.rexxla.org/
[14] http://rexxla.org/
[15] http://www.rexxla.org/rexxla/join.html
[16] http://groups.google.com/group/comp.lang.rexx/topics?hl=en

Brief Overview

ooDialog is a *framework* that aids ooRexx programmers in adding graphical elements to their Rexx programs. The framework provides the base infrastructure, through a number of classes, that the programmer builds on to quickly produce Windows dialogs. This book is a reference to the ooDialog classes, methods, and utilities that make up the base infrastructure.

In general, the ooDialog framework simply provides the Rexx programmer with an interface to the Windows API, and primarily to the part of the API that deals with dialogs and dialog controls. In almost all cases, the behavior of the dialog and its controls is dictated by the Windows API. ooDialog has very little control of this. While this document strives to be complete enough that a Rexx programmer, knowing very little of the Windows API, can effectively write graphical programs in Rexx, it can never be as comprehensive as the actual Microsoft documentation. Therefore, the Rexx programmer that needs, or desires, to go beyond the basic dialog and dialog behavior, will benefit greatly by consulting the MSDN *documentation*.

1.1. Getting Started

The ooDialog documentation should be divided into two parts - a tutorial and a reference. In the original documentation accompanying IBM's Object Rexx, the documentation **was** in two parts. Unfortunately, the tutorial portion mostly described how to use the IBM Resource *Workshop*. Because the tutorial section was primarily directed towards using the Resource Workshop, it does not make much sense in the current context.

This book is primarily a reference that describes the classes and methods in detail. There is no tutorial contained within the book. In ooDialog 4.2.0, a new document, the ooDialog User Guide has been started. It does contain a tutorial section, but at this point it is still a work in progress. The User Guide is a good starting point for the newcomer and will help to get started using ooDialog. In addition to the ooDialog User Guide, the sample ooDialog programs that accompany the ooRexx distribution are probably the best additional source of help for learning how to use ooDialog. However, there are also numerous snippets of example code in this book. In addition the getting *help* section of this reference lists a number of resources for the programmer with questions about ooDialog.

1.2. Definition of Terms

A collection of definitions and explanations for terms used in the ooDialog documentation. These terms may not be familiar to the average Rexx programmer.

1.2.1. Client / Nonclient Area

The *client area* of a window is the part of a window where, normally, the window does its active drawing. For a top-level window this is usually where an application displays its output. The title bar, menu bar, window menu, minimize and maximize buttons, sizing border, and scroll bars are referred to collectively as the window's *nonclient* area. The operating system manages most aspects of the nonclient area. The application manages the appearance and behavior of its client area.

Dialogs are top-level windows, and the client area is the area where the dialog draws its controls. The nonclient area is the border, title bar, etc.. For dialog controls, the client area is where the control draws itself. The nonclient area would be the border of the control, if it has one.

1.2.2. Color

Windows supports a method of specifying colors by using an index into a *color palette*, see below. Palette color indexes are limited in number. The typical palette color indexes are 0 (black), 1 (dark red), 2 (dark green), 3 (dark yellow), 4 (dark blue), 5 (purple), 6 (blue grey), 7 (light grey), 8 (pale green), 9 (light blue), 10 (white), 11 (grey), 12 (dark grey), 13 (red), 14 (light green), 15 (yellow), 16 (blue), 17 (pink), 18 (turquoise).

1.2.3. Color Palette

An array that contains color values identifying the colors that can currently be displayed or drawn on the output device.

Color palettes are used by devices that can generate many colors but can only display or draw a subset of them at a time. For such devices, Windows maintains a system palette to track and manage the current colors of the device.

Applications do not have direct access to this system palette. Instead, Windows associates a default palette with each device context. Applications can use the colors in the default palette.

The default palette is an array of color values identifying the colors that can be used with a device context by default. Windows associates the default palette with a context whenever an application creates a context for a device that supports color palettes. The default palette ensures that colors are available for use by an application without any further action. The default palette typically has 20 entries (colors), but the exact number of entries can vary from device to device. The colors in the default palette depend on the device. Display devices, for example, often use the 16 standard colors of the VGA display and 4 other colors defined by Windows.

1.2.4. COLORREF

Windows uses a type named COLORREF to specify colors as RGB values. The RGB color model specifies a color by using a number, 0 to 255, to represent each of the three primary colors, red, green, and blue. A COLORREF is a single number that Windows will interpret as the 3 values for red, green, and blue. Many of the Windows APIs use a COLORREF as an argument when working with colors.

ooDialog provides a number of methods to make it easy for the programmer to create a COLORREF number by specifying the 3 red, green, and blue values separately. Among these methods are the *colorRef* class method of the *Image* class and the *rgb* method of the *CustomDraw* class.

1.2.5. Conventional Hexadecimal Format

There are a number of methods in the ooDialog framework that have an argument that can be in numeric format, a whole number, or in a *conventional hexadecimal* format. For the purposes of this documentation this hexadecimal format is defined to be a Rexx string that begins with "0x" followed by a maximum of 16 characters, which are only characters that represent hexadecimal numbers. I.e., "0" through "F". The following is meant to clarify this:

```
-- Acceptable:
"0xffff"
"0XFFFF"
"0x000012aB"
"0xFFFF0000aaaa9999"
"0x0"

-- Incorrect:
" 0xffff"               -- leading space
"0XFZFF"                -- Z is not a hexadecimal symbol
"000012ab"              -- second character must be x
"0x0123456789ABCDEF1"   -- 17 characters
```

```
"00x0"                 -- second character must be x, not 0
```

1.2.6. #define Statement

Define statements are often used in the C and C++ languages to define symbolic names for numerical values. Because of this, it is common in Windows programs with dialogs to define symbolic names for resource IDs. Most Windows resource editors use symbolic IDs, (some to a limited degree, others exclusively.) Often the define statements are put in a header file so they are available both to the resource compiler and to the program code. The defines take the form of: **`#define symbolicName numericValue`** as in this example:

```
#define    ID_PUSHBUTTON1  413
#define    ID_EDIT1        511
#define    ID_LISTBOX1     602
```

1.2.7. Device Context

A device context is associated with all windows that appear on the screen, such as a dialog or a dialog control. It is a drawing area managed by a window. A device context stores information about the graphic objects that are displayed, such as bitmaps, lines, and pixels, and the tools used to display them, such as pens, brushes, and fonts. A device context can be acquired for a dialog or a dialog control. It must be explicitly freed when the text or graphic operations are completed.

1.2.8. Dialog Icon

The term *dialog icon* is used in this documentation to refer to the icon that is displayed in the left hand corner of the title bar of a dialog. In Windows this is often called the *application* icon. The dialog icon is also used for the Task Bar display and in the AltTab task switcher application.

The dialog icon for a specific dialog can be set when the dialog is run using one of the execute methods. See the *execute* or *popup* methods for example. ooDialog provides four icon images for use in dialogs. Other, custom, icons can be used by including the icon in a binary (compiled) resource, a resource script, or by using the *addIconResource* method of the UserDialog. The following table shows the pre-defined symbolic IDs of the icon images provided by ooDialog. The symbolic ID should always be used in case the numeric value is changed in the future. In addition, the programmer should avoid using any of the *pre-defined* symbolic IDs reserved by ooDialog. The IDI_DLG_DEFAULT is a fifth symbolic ID that represents the default dialog icon. This ID can always be used where a dialog icon ID is needed.

Table 1.1. ooDialog Supplied Icons

Description	Symbolic ID
The default, the letters OOD	IDI_DLG_OODIALOG
Dialog box image	IDI_DLG_APPICON
Fancier dialog box image	IDI_DLG_APPICON2
The ooRexx image	IDI_DLG_OOREXX
IDI_DLG_DEFAULT	IDI_DLG_DEFAULT

1.2.9. Dialog Unit

Dialog box templates contain measurements that define the size and position of the dialog box and its controls. These measurements are device independent. This allows a single template to be used to create the same dialog box for all types of display devices. Using device independent measurements allows a dialog box to have the same proportions and appearance on all screens despite differing resolutions and aspect ratios between screens.

These measurements are called dialog template units, often shortened to just dialog units in this documentation.

The following paragraph in italics, which has been the sole documentation of dialog units in the ooDialog documentation prior to version 4.0.0, is unfortunately incorrect. The value of a dialog unit is dependent on the font actually used in the dialog, not on the system font. The statements below were probably true in very early versions of Windows when every dialog used system 8 pt font. Today it is highly unusual for a dialog to use system 8 pt font. The factorX and factorY values are calculated incorrectly. These values are only correct if the dialog is using system 8 pt font and are incorrect for a dialog using any other font.

There is a horizontal and a vertical dialog base unit to convert width and height of dialog boxes and controls from dialog units to pixels and vice versa. The value of these base units depend on the screen resolution and the active system font; they are stored in attributes of the UserDialog class.

```
xPixels = xDialogUnits * self~FactorX
```

Note that in the above line of code **`xPixels`** will be *inaccurate*.

1.2.10. Handle

A unique reference to a Windows object assigned by the system. It can be a reference to a dialog, a particular dialog control, a window, or a graphic object (pen, brush, font). Handles are required for certain methods. A handle is an opaque type, the Rexx programmer need not be aware of the specific format of a handle. The ooDialog framework provides methods that return handles and methods for retrieving handles from the operating system. When a method requires a handle as an argument, the Rexx programmer needs to obtain the handle from one of those provided methods.

1.2.11. Header File

A common practice when programming applications in Windows that use dialogs and dialog resources is to place symbolic defines in a separate file. These files often have a .h extension and are usually called header files. Windows resource editors often manage a header file for the symbolic IDs automatically. (For instance Microsoft's dialog editor creates, writes, and reads the resource ID header file completely on its own. The user does not need to take any action other than including the file in her program.)

1.2.12. Modal and Modeless Dialogs

Dialogs are executed in two basic ways. A *modal* dialog blocks keyboard and mouse input to all other windows started by the program. The user can not switch to another window in the program without closing the modal dialog. In ooDialog this is often all other dialogs started by the program. A *modeless* dialog operates independently of the other dialogs in the program. The user can switch away from a modeless dialog and work with any of the other dialogs in the program.

Be aware that the original developers of ooDialog choose to not implement true modal dialogs. Rather, they implemented a strategy where all dialogs are created as modeless dialog and the ooDialog framework keeps track of the last dialog executed. This *last executed* dialog is then manually disabled to mimic the behaviour of true modal dialogs. While this strategy is usually sufficient, it will sometimes cause the wrong dialog to be disabled.

In the ooDialog framework use the *execute* method to create modal dialogs and the *popup* or *popupAsChild* methods to create modeless dialogs.

1.2.13. Pixel

Individual addressable point on the monitor (screen or display.) Pixels are whole numbers. Ancient VGA screens supported 640 by 480 pixels, SVGA screens supported higher resolutions, such as 800 by 600, 1024 by 768, and up. Modern displays support much higher resolutions. 1600 by 1200 is common and displays with 2560x1600 are available. Pixel values start at the top left corner of the main display, with that corner being (0,0). Prior to dual monitor capabilities, pixel values were always non-negative. On a dual monitor system that is no longer true, depending on the virtual position of the secondary monitor. If it is to the left or above the primary monitor negative pixel values are possible.

1.2.14. Resource Editor

Resource editors are visual tools used to create a dialog *template* in a text file. Visual resource editors provide a WYSIWYG (what you see is what you get) environment to design dialogs. The editor manages the size and positioning of a dialog and its controls. The user can drag and drop controls where they are wanted and use the mouse to size the dialog and controls. Resource editors simplify the process of designing the look of a dialog and reduce the amount of trial and error design inherent in using the *UserDialog* class.

1.2.15. Resource ID

A resource ID is the identification number of a dialog resource. There are several different types of dialog resources, menus, dialog controls, and bitmaps, to name a few. You assign IDs when you create the resource definition for your dialog. An ID can be either numerical (for example, 1) or symbolic (for example, "IDOK").

IDs must be unique for each resource of the same type. Although two resources of different types may have the same ID, when using *symbolic* IDs within the ooDialog framework it is advisable to give all resources unique numerical IDs.

1.2.16. Resource Script

Resource script files are plain text files usually produced by a resource editor. The files generally have a file extension of ".rc", but an extension of ".dlg" is used by some resource editors. The text of a resource script defines a dialog *template*. The format of the text is defined by Microsoft and public knowledge. The format is easily parsable by computer software and is used by resource compilers to produce a compiled (binary) file containing the dialog template(s) defined in the script file. The ooDialog framework can parse a resource script file and dynamically produce a dialog template in memory. This is the basis of how a *RcDialog* works.

1.2.17. Screen / Client Coordinates

Points on the screen are described as x and y coordinate pairs (x,y). The x coordinates increase to the right, y coordinates increase towards the bottom. *Screen* and *client* coordinates are used to distinguish the origin (0,0) of the coordinate. For a screen coordinate, the origin is the upper left corner of the primary display device, typically a monitor, and usually called the screen. Client coordinates on the other hand have an origin of the upper left corner of the *client area* of the window. Both screen and client coordinates are always given in the device unit of the display, which for all practical purposes is a *pixel*.

1.2.18. Symbolic ID

A symbolic name is a constant symbol that uniquely identifies a specific entity in a program. Defining a symbolic name for each numeric resource ID is often done in programs that work with resource IDs.

The symbolic name is then used where ever a numeric resource ID is needed. Symbolic names are easier to remember than numeric IDs and can make the code easier to understand.

Symbolic names are most often used in compiled programs, where a preprocessor replaces each occurrence of the symbolic name in the code with its numeric value before the code is compiled. Symbolic names are less often used in interpreted languages because there is no preprocessor step where substitutions can be made. However, ooDialog provides a robust and useful *mechanism* for using symbolic IDs in ooDialog programs. Programmers wishing to use symbolic resource IDs in their programs should be familiar with this *mechanism*.

1.2.19. SystemColor

The Windows operating system maintains a table of system colors for each display element. Display elements are the parts of a window and the display that appear on the system display screen. The user can customize the color of each display element, so each element is assigned a unique numeric and symbolic ID. Referring to one of the system colors by its ID allows the programmer to use the correct color of a display element without having to know if the user has customized the color for that element or not.

The following table lists the numeric and symbolic keyword for each display element. In the ooDialog framework, for methods that work with the system colors, like *setSysColor* or *setControlSysColor*, the system color can be specified using either its numeric ID or its keyword. Note that the operating system symbol for each keyword actually is prefaced by COLOR_. I.e., the actual symbol for 3DDKSHADOW is COLOR_3DDKSHADOW, for 3DFACE the symbol is COLOR_3DFACE, etc.. This table lists the ooDialog keyword.

Table 1.2. System Color Elements

ID	Symbol	Element
21	3DDKSHADOW	Dark shadow for three-dimensional display elements.
15	3DFACE	Face color for three-dimensional display elements and for dialog box backgrounds.
20	3DHIGHLIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
20	3DHILIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
22	3DLIGHT	Light color for three-dimensional display elements (for edges facing the light source.)
16	3DSHADOW	Shadow color for three-dimensional display elements (for edges facing away from the light source).
10	ACTIVEBORDER	Active window border.
2	ACTIVECAPTION	Active window title bar. Specifies the left side color in the color gradient of an active window's title bar if the gradient effect is enabled.
12	APPWORKSPACE	Background color of multiple document interface (MDI) applications.
1	BACKGROUND	Desktop.
15	BTNFACE	Face color for three-dimensional display elements and for dialog box backgrounds.

ID	Symbol	Element
20	BTNHIGHLIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
20	BTNHILIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
16	BTNSHADOW	Shadow color for three-dimensional display elements (for edges facing away from the light source).
18	BTNTEXT	Text on push buttons.
9	CAPTIONTEXT	Text in caption, size box, and scroll bar arrow box.
1	DESKTOP	Desktop.
27	GRADIENTACTIVECAPTION	Right side color in the color gradient of an active window's title bar if the gradient effect is enabled. ACTIVECAPTION specifies the left side color.
28	GRADIENTINACTIVECAPTION	Right side color in the color gradient of an inactive window's title bar. INACTIVECAPTION specifies the left side color.
17	GRAYTEXT	Grayed (disabled) text. This color is set to 0 if the current display driver does not support a solid gray color.
13	HIGHLIGHT	Item(s) selected in a control.
14	HIGHLIGHTTEXT	Text of item(s) selected in a control.
26	HOTLIGHT	Color for a hyperlink or hot-tracked item.
11	INACTIVEBORDER	Inactive window border.
3	INACTIVECAPTION	Inactive window caption. Specifies the left side color in the color gradient of an inactive window's title bar if the gradient effect is enabled.
19	INACTIVECAPTIONTEXT	Color of text in an inactive caption.
24	INFOBK	Background color for tooltip controls.
23	INFOTEXT	Text color for tooltip controls.
4	MENU	Menu background.
29	MENUHILIGHT	The color used to highlight menu items when the menu appears as a flat menu. The highlighted menu item is outlined with HIGHLIGHT.
30	MENUBAR	The background color for the menu bar when menus appear as flat menus. However, MENU continues to specify the background color of the menu popup.
7	MENUTEXT	Text in menus.
0	SCROLLBAR	Scroll bar gray area.

ID	Symbol	Element
5	WINDOW	Window background.
6	WINDOWFRAME	Window frame.
8	WINDOWTEXT	Text in windows.

1.2.20. System Error Code

The term *system error code* refers to an error code set by the Windows operating system when an API fails. ooDialog provides an interface to the Windows APIs and when an error is detected many of the ooDialog methods have some means of conveying the system error code to the programmer. The ooDialog programmer can look up the meaning of a system error code in the MSDN *documentation* to understand better the cause of a failure.

ooDialog provides the *.systemErrorCode* environment entry as a means for the Rexx programmer to determine the value of the system error code after a method has executed. Note that not all of the Windows APIs set the system error code. And, likewise, not all of the ooDialog methods set the **`.systemErrorCode`**.

1.2.21. Windows Documentation

The term *Windows documentation* is used throughout the ooDialog reference to refer to the Windows Operating System documentation provided by Microsoft. The documentation is called the **MSDN Library**. The library is provided online for anyone to access. In addition, since May 2006, Microsoft has also provided free of charge the ISO images of the library installation program. Anyone can download the ISOs, burn them to a CD and install the library locally on their system.

It is not necessary for the ooDialog programmer to know or understand the underlying Windows API that ooDialog is built on. However, as programmers write more sophisticated ooDialog applications, it may prove helpful to look up certain details in the MSDN Library. The information below is provided to help the ooDialog programmer locate the MSDN Library, if they would like to. All things on the Internet change. The URLs listed here are accurate at the time of this writing.

The online MSDN Library is currently located at:

http://msdn2.microsoft.com/en-us/library/default.aspx.

Directions to the downloadable ISO images of the MSDN Library have been posted on this blog entry:

http://blogs.msdn.com/robcaron/archive/2006/07/26/678897.aspx

A Google search using: **`"Rob Caron" General Downloads MSDN Library`** should also turn up the blog entry.

1.2.22. Windows Platform SDK

The *Windows Platform SDK* is provided free of charge by Microsoft. The SDK is not needed to write ooDialog programs. However, combining the use of the documentation in the MSDN Library with the SDK allows very sophisticated ooDialog programs to be written. In general, the ooDialog framework takes care of the low-level details needed to work with the Windows API. However, there are a few generic ooDialog methods that provide direct access to the Windows API.

As an example, the *addUserMsg* method allows the programmer to connect any Windows message sent to a dialog to a method in his ooDialog class. To use this method, the programmer would go to the MSDN library to look up details on the message and message parameters he is interested in. He would then use the Platform SDK to determine the numeric value of the Windows message and possibly the numeric values of its parameters.

This link provides some good information on the Platform SDK in general and also points the reader to where to get a SDK.

http://en.wikipedia.org/wiki/Platform_SDK

Again, note that it is not at all necessary to obtain, or understand details concerning, the Platform SDK. This information is provided for those programmers that have reached the point where they think a method like **`addUserMessage`** might help them and need some direction as to how to go about using it.

1.3. Common Concepts

Many concepts and behaviors in ooDialog, and statements about ooDialog are general in nature. This section gathers up this information in one place. Rather than repeating this information in every method or class description to which it applies, the author will assume that the reader understands that the information is always applicable unless specifically stated otherwise.

1.3.1. Deprecated

PROVIDE TEXT

1.3.2. Dialog Template

The *underlying* dialog seen by the user is created by the operating system from a dialog template in memory. The template describes the size and position of the dialog and all of its controls. The template also contains modifiers that control the style, behavior, and attributes of the dialog and its controls. To make designing dialogs easier, Windows supports the concept of a textual representation of the dialog template. The textual representation can then be translated by software tools to the binary form of the template needed by the operating system.

ooDialog uses three basic constructs that allow the Rexx programmer to supply the dialog template. The programmer can use a binary resource, a resource script, or create the template dynamically in the program code. A binary (compiled) dialog template is stored in a DLL, (usually a .dll file.) The programmer subclasses a *ResDialog* to use a binary resource. Resource scripts, (usually a .rc file) supply the dialog template in a text file. The programmer subclasses a *RcDialog* to create a dialog from a resource script. To create a dialog template dynamically in the program code, the programmer subclasses a *UserDialog* and then uses the *create...* methods of that class to create the dialog template. The **`UserDialog`** object translates the program statements into the in-memory dialog template required by the operating system. The **`RcDialog`** object parses the resource script and converts the resource script statements into an in-memory dialog template by invoking the proper methods of its superclass, the **`UserDialog`**.

1.3.3. Events

The ooDialog framework facilitates the use of a type of programming often called *event driven programming*. In event driven programming, the program usually does some initial set up and then sits in some type of loop waiting to be signaled that an event just happened. The Windows graphical windowing system is designed to be programmed this way. The Windows operating system uses *messages* to notify each window in the system of events specific to that window. Typically events are generated by the user. For instance, clicking a button, typing a key, moving the mouse, all generate events. The operating system notifies the window with the input focus of those events by sending *message*s to the window. Note that some events are generated by the operating system itself. For instance, when the user moves a window that uncovers a portion of a window beneath it, the operating system will send a message to the underlying window notifying it that it needs to redraw the uncovered portion.

Once the basic set up for an ooDialog program is done, the dialog object basically sits there waiting for an *event* of interest to happen. When the event happens, the program responds by taking some action. The programmer decides what events are of interest and uses methods provided by the ooDialog framework to *connect* a method in the dialog object to the event notification. The majority of the event connection methods are part of the *EventNotification* class. The connected methods are often called *event handlers* because the code in the method handles the event.

Directly Reply

Event notification messages in Windows fall into two groups, messages where the reply is ignored and messages where the reply is significant. Prior to the introduction of the C++ *native* APIs, there was **no way** in ooDialog to *directly* reply to the notification message. This placed a severe restriction on ooDialog programs. Many of the features of the operating system could not be used with this restriction. For instance, when a user selects a new tab in a *Tab* control, the operating system sends a SELCHANGING event notification before the selected tab is changed. The programmer can allow or prevent the change by replying true or false to the notification message.

Without the ability to reply directly to the notification, the ooDialog programmer could not take advantage of the SELCHANGING notification. The introduction of the C++ native APIs in ooRexx 4.0.0 removed this restriction. Beginning in ooDialog 4.2.0, the event handling methods in the Rexx dialog object can be directly invoked from the Windows message processing loop. This allows the Rexx dialog object to reply directly to the notification message.

In addition, the underlying Windows message processing loop provides a form of synchronization in Windows applications. Within the loop, a Windows application receives a message, processes it, then receives the next message, processes it, and so on. In older ooDialog programs this synchronization was lost because ooDialog put the received message on a queue, received the next message, put it on the queue, and continued. This meant that many messages could be received in the processing loop before a single message was process by the ooDialog program. This loss of synchronization caused ooDialog applications to perform poorly.

The ability to directly reply to event notifications greatly extends the power of the ooDialog framework. However, it also changes how the ooDialog programmer must write his event handlers. In particular, the event handler must return in a timely manner. This is discussed more *fully* in the **`EventNotification`** class documentation.

1.3.4. factorX / factorY

The *factorX* and *factorY* attributes of the *dialog* object were intended to provide a way to convert between *pixels* and dialog *units*, and vice versa. Although their values may have been correct when ooDialog was originally *designed*, in almost all cases the values are now incorrect. The method used to calculate the ratio between dialog units and pixels is not correct.

Unfortunately, many of the methods in the ooDialog framework use *factorX* and *factorY* to convert between pixels and dialog units. This in turn makes all of those methods inaccurate. These methods are all marked as being inaccurate. There is almost always no reason to convert back to dialog units from pixels. Once the underlying dialog has been created, pixels should be used. Each inaccurate method in the framework has a corresponding method that uses pixels instead of dialog units.

1.3.5. IBM Resource Workshop

The IBM Resource Workshop was a visual *resource editor* included with IBM Object Rexx. The Resource Workshop could not be contributed to the open source community and is therefore not a part of the Open Object Rexx project.

Note, there is **no loss** of functionality in ooDialog because of the absence of the Resource Workshop. The Windows resource format is well understood and there are any number of free or inexpensive resource editors that do a better job of designing dialogs than the Resource Workshop did. (The Resource Workshop was a 16-bit application with limited capacity for the newer features in the Windows user interface.) ooDialog works fine with dialogs designed by any modern resource editor.

1.3.6. Numbers in ooDialog:

Numbers in ooDialog are **always** whole numbers, unless specifically stated otherwise. Except in very rare cases, the Windows API that ooDialog provides access to, only deals with whole numbers. Pixels, positions, sizes of fonts, coordinates on the screen, etc., are all expressed as whole numbers only. Numerical arguments to methods in ooDialog must always be whole numbers, unless the documentation specifically notes that the method accepts fractional numbers for the argument.

1.3.7. Predefined Symbolic IDs

The symbolic IDs in the following table are pre-defined by ooDialog and always available to the programmer. They are placed in the either the *constDir* attribute when an instance of a dialog class is created, or in the global *.constdir*. Where they are placed is dependent on the *useGlobalConstDir* of the **`.constDir`**. All symbolic names after IDC_STATIC in the table refer to resources bound to oodialog.dll for general use by the ooDialog programmer.

To allow for future expansion, the ooDialog programmer should consider the resource IDs of 1 through 50 as reserved for ooDialog. Programmers can avoid conflicts by using IDs greater than 50 for resource IDs they assign in their programs.

Table 1.3. Symbolic IDs Predefined by ooDialog

Symbolic ID	Numeric ID or Symbol	ResourceType
IDOK	1	ID for button controls
IDCANCEL	2	ID for button controls
IDABORT	3	ID for button controls
IDRETRY	4	ID for button controls
IDIGNORE	5	ID for button controls
IDYES	6	ID for button controls
IDNO	7	ID for button controls
IDCLOSE	8	ID for button controls
IDHELP	9	ID for button controls
IDTRYAGAIN	10	ID for button controls
IDCONTINUE	11	ID for button controls
IDC_STATIC	-1	ID for static controls
IDI_DLG_OODIALOG	IDI_DLG_OODIALOG	Icon image
IDI_DLG_APPICON	IDI_DLG_APPICON	Icon image
IDI_DLG_APPICON2	IDI_DLG_APPICON2	Icon image
IDI_DLG_OOREXX	IDI_DLG_OOREXX	Icon image
IDI_DLG_DEFAULT	IDI_DLG_DEFAULT	Icon image

The symbolic IDs in the table following IDC_STATIC are the IDs of some generic *resources* that are bound to the oodialog.dll file. They can be used in any ooDialog program and are accessed using the *ResourceImage* class. Programmers should always use their symbolic ID rather than their numeric ID in case the numeric value changes in future versions of ooDialog.

1.3.8. Required Common Control Library (Comctl32) Version

The dialog control windows used in dialogs, List-Views, Edit, Tree-Views, etc., are supplied by Microsoft in the common controls library. This is a DLL with the name comctl32.dll. Every version of Windows is supplied with a common controls library. However, Microsoft has updated the library a number of times to provide enhanced functionality and improved features

Each new version of the library is backwards compatible with previous versions, but, it will contain features not available in older versions. For instance, some of the List-View extended *styles* are only available with a 6.0, or later, version of the common controls library. ooDialog can only provide the features available in the version of the common controls library on the system ooDialog is running on.

Therefore, an ooDialog program running on a Windows 2000 machine will not have available some of the features that are available when ooDialog is running on a XP service pack two system. The DlgUtil class provides a method, *comCtl32Version* that allows the programmer to determine the exact version of the common controls library that ooDialog is using. In the documentation for the ooDialog dialog control classes, features that are not available in all versions of the common control library are noted. The minimum version of the library that is needed is listed. In general, at this time, all features of ooDialog are available on Windows XP or later. This may change in the future as Vista has common control features not available on XP.

1.3.9. Rectangle Coordinates

The ooDialog framework, and Windows itself, often use a *Rect* object to specify coordinates of a window, or a portion of a window. However, there are two different ways the rectangle is used. For purposes of this documentation, the two types of rectangle are defined as a **bounding** rectangle and a **point/size** rectangle.

Bounding Rectangle:

In a *bounding* rectangle, the members of the `Rect` object define the upper left point of the rectangle and the lower right point of the rectangle. That is, the *left* and *top* attributes of the `Rect` object are the (x,y) coordinates of the upper left corner of the rectangle and the *right* and *bottom* attributes are the (x1,y1) coordinates of the bottom right corner of the rectangle.

In this usage, the width of the rectangle is derived by subtracting the *left* attribute from the *right* attribute and the height of the rectangle is derived by subtracting the *top* attribute from the *bottom* attribute.

Point / Size Rectangle:

In a *point / size* rectangle, the members of the `Rect` object define the upper left point of the rectangle and the size of the rectangle. That is, the *left* and *top* attributes of the `Rect` object are the (x,y) coordinates of the upper left corner of the rectangle. The *right* attribute of the `Rect` object is the width of the rectangle. The *bottom* attribute is the height of the rectangle

In this usage, the x coordinate of lower right corner of the rectangle is derived by adding the *left* and *right* attributes. The y coordinate of the lower corner of the rectangle is derived by adding the *top* and *bottom* attributes.

1.3.10. Required Windows Version

The required Windows version is similar to the required Common Control *Library* library version. Later versions of the Windows operating system have dialog and dialog control features not available in earlier versions of the operating system.

For instance, the *MonthCalendar* class has a number of methods that are only available on Vista or later. ooDialog can only use the features available on the system on which it is executing. If a method is invoked that is not available on the current operating system, a syntax conditions is raised. Any method not available on all the Windows versions which ooRexx supports, have the minimum required Windows version noted in their documentation.

The *OS* class supplies methods that allow the programmer to determine exactly which operating system version the program is currently executing on. For programs that need to run on all versions of Windows, the programmer must either avoid using methods not available on all versions, or test for the current version and provide an alternative code path dependent on that version. The *isAtLeastVista* method of the *OS* class provides an example of this.

1.3.11. Underlying Dialog Creation

The dialog and dialog control objects in ooDialog represent the dialogs and controls users see on their screens. This documentation often refers to the underlying dialog or the underlying control. These statements refer to the dialog or control created by the operating system, the objects the user sees on the screen. The operating system controls what these underlying objects can, and can not, do. One thing that it is sometimes difficult for the Rexx programmer to grasp is that ooDialog can not alter the behavior or appearance of these objects in ways not allowed by the operating system. The Windows API provides a broad number of ways to customize the appearance and behavior of dialogs and controls. But, ooDialog and the Rexx programmer are restricted to those customizations provided by the operating system.

Another concept that is often hard to grasp is that many of the methods of the ooDialog dialog and control objects can only be used after the underlying Windows dialog has been created. In general this means in the *initDialog* method or later in the life cycle of the dialog. In particular, the *defineDialog* method of the *UserDialog* class executes before the underlying dialog is created. Therefore, any method that requires the underlying dialog to have been created can not work in the *defineDialog* method. In earlier versions of ooDialog, in general, if the programmer invoked a method requiring that the underlying dialog was created, the error was simply ignored. But, the method had no effect.

From the 4.0.0 release of ooRexx and on, the goal in ooDialog is to raise a syntax condition when a method requiring the underlying dialog is invoked and the underlying dialog does not exist. This goal is being implemented over time and may not yet be completed.

1.3.12. Undocumented Items

ooRexx is open source and anyone can peruse the source to see all functions, classes, and methods of the ooDialog framework. Any of these items that are undocumented should not be used by the Rexx programmer. If the programmer does use any undocumented features in the framework, he does so at his own risk. The framework strives to be backwards compatible, but only for documented features. Undocumented features in the framework are intended for internal use only and are subject to change, or even removal from the framework.

In addition, previous versions of the ooDialog documentation, documented some features, but added some form of the caveat: *for internal use*. The prudent programmer would not use any feature documented for internal use. These features are also subject to change.

1.3.13. Using Symbolic IDs in ooDialog:

ooDialog allows programmers to use *symbolic* resource IDs in their programs. The symbolic ID can be used for any argument in any method that requires a resource ID. Although this is mostly transparent to the programmer, if the programmer is going to use symbolic IDs, he should understand the basics of the mechanism allowing the use of symbolic IDs.

Resolving Symbolic IDs:

ooDialog uses a **`.Directory`** object that consists of indexes that are symbolic names. The value for the item at each index is the numeric value of the symbolic name. This gave rise to the object name of **`constDir`**, which probably refers to a directory of constants (symbols.) When an argument in a method is a resource ID, ooDialog first checks to see if the argument is a whole number. If so, it simply uses the number. When the argument is not a number, ooDialog uses the argument to do a look up in the constant directory object. If the argument matches an index in the object, the value of the index is used as the numeric value of the resource ID. When the argument does not match an index in the constDir, the method fails.

constDir Attribute:

Originally the constDir was implemented as an *constDir* of the dialog object. Each dialog object then has its own constDir, which was used to resolve symbolic IDs where needed in the methods of the dialog. Each dialog control has a reference to its owner dialog, allowing the dialog object's constDir to be used to resolve symbolic IDs for the methods of the control. This mechanism exists unchanged in the current ooDialog framework.

However, as enhancements to ooDialog started to be made in version 4.0.0 and later, it became apparent that the constDir attribute was not adequate. Classes such as the *Menu* classes and *ResourceImage* class also used resource IDs, but objects of these classes are independent of any dialog object. Without access to a dialog object, there was no way in those classes to use symbolic IDs. To fix this problem, ooDialog 4.2.0 added the global **`.constDir`** to the mechanism for using symbolic IDs.

Global .constDir object:

The global *.constdir* works in the same way as the dialog object's constDir attribute. When a symbolic ID needs to be resolved, it can be looked up in the global **`.constDir`**. The integration of the global **`.constDir`** into the mechanism for resolving symbolic IDs is done by allowing the programmer to specify how, or even if, the **`.constDir`** should be used. The four choices are *only*, *first*, *last*, or *never*. The programmer specifies which of these strategies is to be used through one of the methods of the *.application*. (The **`.application`** object is an instance of the *ApplicationManager* class.)

Global .constDir Usage Strategies:

The four strategies work this way.

Only:

With this strategy, symbolic IDs are only placed in the **`.constDir`** and only looked up through that object. When a new dialog object is instantiated, rather than assign a newly instantiated **`.Directory`** object to its constDir attribute, the global **`.constDir`** is assigned to the attribute. This is by far the most efficient way to use symbolic IDs in an ooDialog program.

First:

When the ooDialog framework needs to resolve a symbolic ID, it will first try to resolve it using the **`.constDir`**. If that fails, it will then try to resolve it using the constDir attribute. If that then fails, the method will fail.

Last:

This is the reverse of the use *first* strategy. The ooDialog framework first tries to resolve a symbolic ID in the constDir attribute. If that fails, the **`.constDir`** is tried.

Never:

The **`.constDir`** is never used in the program. To preserve backwards compatibility, this is the default.

Adding Symbols:

A few symbolic IDs are *pre-defined* by ooDialog and are always present in the constant directory. These symbolic IDs can be used in any ooDialog program. Other than the pre-defined IDs, the programmer must add symbolic IDs in order to use them in a program.

The easiest way to add symbols is to have the ooDialog framework add the symbols itself from a file. Whenever ooDialog parses a resource *script* or a header *file*, it automatically adds any symbol *definitions*s it finds to a constant directory. Which constant directory is dependent on the global **`.constDir`** usage strategy in effect.

Resource scripts are parsed whenever a *RcDialog*, or subclass, is instantiated. In addition, all the ooDialog dialog classes accept a header file as an optional parameter when a new instance of a dialog object is created. (See for example the **new** method in the *new* object or the *new* class.) If the programmer supplies the optional header file argument, the ooDialog framework automatically parses the file and adds all defined symbols it finds in the file to a constant directory. Again, to be clear, which constant directory the symbol is placed in is dependent on which **`.constDir`** *strategy* the programmer has elected to use.

In practice, most resource scripts are written by *resource editor*s and place the symbol definitions in a separate header file. Therefore having the ooDialog framework read and parse a header file is the most practical way to add symbols to a constant directory. In addition to using the optional header file argument in the *new* method of a dialog, the ooDialog framework provides some additional methods for reading a file and adding the symbol definitions to a constant directory. The dialog object has the *parseIncludeFile* method. The *ApplicationManager* class has several methods, which are accessed through the *.application*. These are: the *useGlobalConstDir*, *addToConstDir*, *setDefaults*, and *parseIncludeFile* methods.

Of course, symbolic IDs can also be added directly in the program as the following code snippet shows:

```
::method init
  forward class (super) continue

  self~constDir[IDC_GB] = 101
  self~constDir[IDC_CB_REGINA] = 107
  self~constDir[IDC_CB_REGINALD] = 111
  self~constDir[IDC_CB_OOREXX] = 115

...

::method defineDialog

  self~createGroupBox(IDC_GB, 10, 20, 150, 90, "BORDER", "Pick an interpreter")
  self~createCheckBox(IDC_CB_REGINA, 30, 40, , , "GROUP", "Regina")
  self~createCheckBox(IDC_CB_REGINALD, 30, 60, , , , "Reginald")
  self~createCheckBox(IDC_CB_OOREXX, 30, 80, , , , "ooRexx")

...

::method ok

  oorexxCB = self~newCheckBox(ID_CB_OOREXX)
  if oorexxCB~checked then
    say "You picked the right interpreter."
```

Classes Requiring the Global .constDir for Symbolic IDs:

As explained previously, the dialog object's *constDir* attribute allows programmers to use symbolic IDs for arguments requiring resource IDs in the methods of the *dialog* object and the dialog *control* object. However, the ooDialog framework contains other classes with methods requiring resource IDs. The *Menu* classes and the *ResourceImage* class are examples of these classes. If the programmer wishes to use symbolic IDs in classes other than the dialog and dialog control classes, then the global **`.constDir`** **must** be used.

Programmers that do not want to use the **`.constDir`**, **must** use numeric resource IDs for any argument requiring a resource ID in any of the classes other than the dialog and dialog control classes.

Global .constDir Pros:

Using the global **`.constDir`** is by far the most efficient way to use symbolic IDs in ooDialog programs.

Using the global **`.constDir`** allows large applications with many dialogs to read the symbol file one time only. When using the dialog object's **`constDir`** attribute, the symbol file would need to be constantly re-read, once for each dialog.

Using the global **`.constDir`** allows the programmer to use symbolic IDs for any method in any object that requires a resource ID.

Global .constDir Cons:

The **`.constDir`** does have some restrictions. These restrictions may be a reason for some programmers to prefer to not use the **`.constDir`**. Depending on one's point of view, many of the cons could be viewed as pluses.

The indexes in the **`.constDir`** are case sensitive. However, this allows symbol resolution to be more efficient and faster. Case sensitivity is not a problem if a distinctive naming scheme is used for symbolic IDs where the symbol names would never be used for variable names, and the symbol names are never quoted.

If a symbolic name can not be resolved, a syntax condition is raised. However, the only reason a symbol would not be resolved is if the programmer used an incorrect symbol. The raised syntax condition would alert the programmer to an error in the program early in the development cycle.

Since all symbolic IDs are in one constant directory, duplicate symbol names can not be used in different dialogs, unless the duplicate symbol names are assigned a single numeric value. (In which case they are not actually duplicates.) For example, take a program that has two different dialogs in it, each with an edit control in the dialog. If the programmer's habit is to use a symbol of IDC_EDIT for edit controls, there could be a conflict if the edit control in the first dialog, had a define like this: **`#define IDC_EDIT 100`** and the edit control in the second dialog had a define like this: **`#define IDC_EDIT 200`**. The solution to this is too either use two different symbols for the two edit controls, or use the same resource ID number for both edit controls.

1.3.14. Window Messages

In the Windows operating system, not surprisingly, most everything is a window. In a nutshell, the Windows operating system works by routing and sending *messages* to these windows. Each window has a *message processing loop* where the window waits for a message to arrive, processes it, and then waits for the next message. Each window message has an unique whole number ID that gives meaning to the message. When a message is sent to a window, it is sent with two arguments. The first argument is of a type called WPARAM and the second is a type called LPARAM. These types are opaque and the value of each is dependent on the specific message. Either or both of the arguments may have no meaning for the specific message. The window always returns a value, another opaque type called LRESULT, which again may have no meaning for the particular message.

In general, most of what ooDialog does is done by sending window messages to the operating system windows. This is designed to allow the Rexx programmer to use dialogs without understanding any of the details of window messages. However, the ooDialog framework provides a few generic methods that send window messages to the underlying operating system windows. These methods, in contrast to most methods, can not be used by the Rexx programmer without some knowledge of the window message being sent. The methods are clearly marked. The programmer would need to consult the Windows *documentation* to understand what both the WPARAM and LPARAM arguments must be, and to understand what the LRESULT return, if any, means. In addition, the programmer would need to determine the numeric value of the window message ID. This could be done using a Windows platform *SDK*, or perhaps through a Google search.

These generic send messages methods allow Rexx programmers to send any message to any of the underlying windows in their programs. The caveat is that the programmers will have to research the meaning of the messages, their arguments, and their return values themselves.

1.4. History

ooDialog is a Windows only extension to Open Object Rexx and appeared in the first version of IBM's Object Rexx for Windows. This section attempts to give a rough over-view of the history of ooDialog and continues with notes pertaining to the sequential releases of ooDialog past ooDialog 4.1.2

1.4.1. Prehistory Timeline

Object REXX for Windows NT & 95 Interpreter Edition V1.0 was announced very early in 1997. It included ooDialog and could run on Windows 95 and Windows NT 3.5. However, comments within the code and sections of code itself in ooDialog, indicate that at one point ooDialog could run on Windows 3.0 and 3.1.

IBM announced and released several successive versions of Object Rexx for Windows over the following years. For instance, Object REXX for Windows V2R1 was announced March 2001. In May 2004, IBM announced its intention to contribute the source code for Object Rexx to RexxLA, and the ooRexx project was born. It took the best part of a year to work out the details and for the ooRexx developers to get the first version of ooRexx built and released. ooRexx 3.0, the first ooRexx release, took place on March 25th 2005.

From that point, the ooRexx team began regularly releasing new versions of ooRexx, up through the release of ooRexx 3.2.0 on November 5, 2007. After the release of 3.2.0, the ooRexx team concentrated on refactoring the interpreter and on adding a new set of native APIs used to write extensions to the ooRexx interpreter. The new APIs allowed the passing ooRexx objects to the routines and methods used in extensions. Previously, only strings could be passed to external libraries. The refactored ooRexx interpreter and the new native APIs were introduced in ooRexx 4.0.0, released in August 2009.

Up until that release in the middle of 2009, ooDialog remained virtually unchanged. The evolution of the Windows operating system had passed it by. ooDialog was designed for Windows 3.1, or at the latest Windows 95. It still referred to dialog controls such as the list-view as "new" controls. Despite the fact that the list-view control had been in use for over a decade. Some controls were considered "advanced" in ooDialog, while in Windows itself they were considered old news.

Much of the design of ooDialog was based on constraints that existed in Windows 3.1 and Windows 95 that no longer existed in modern operating systems. For instance, the reason for having three separate ooDialog class files, **`OODPLAIN.CLS, OODIALOG.CLS, and OODWIN32.CLS`**, and for providing the simpler **`PlainUserDialog`** class was to have a smaller package that required less system resources for ordinary user interfaces like the standard dialogs. That reason is not as valid in modern times as it was when ooDialog was being developed to run on Windows 3.1. Many of the programming techniques in the ooDialog implementation were also outdated. The conversion of dialog units to pixels and vice versa in ooDialog is a good example. While the technique must have worked in very early versions of Windows, it is completely wrong in Windows XP and later.

The release of the new C++ native APIs in ooRexx 4.0.0 allowed much richer extensions to the Rexx interpreter to be written. Extensions could be implemented that had access to the actual objects instantiated within the interpreter. During the release of ooRexx 4.0.0, the internal work was started to convert ooDialog to use the C++ APIs. This work lead to the complete refactoring of the ooDialog code, and the effort to bring ooDialog up to the level of Windows Vista and Windows 7 got started. New dialog controls were added along with new dialog objects and methods. However, this work was done in the internal development source tree and was not ready for the 4.0.0, 4.0.1, and 4.1.0 releases of ooRexx. Although ooDialog in those releases had some improvements and enhancements, they were minor.

ooRexx 4.1.0 was released in December 2010. At that time, the internal work to convert ooDialog to the C++ APIs and the refactoring of the ooDialog source code was complete, but the refactored ooDialog was not ready for release, primarily because the documentation was not complete. The ooDialog distributed with ooRexx 4.1.0 was essentially the same as the ooDialog distributed with ooRexx 4.0.0, with bug fixes and some small enhancements. A small bug fix version of ooRexx 4.1.1 was released in March 2012. The ooDialog distributed in 4.1.1 is identical to the 4.1.0 ooDialog.

When the work of getting ooRexx 4.1.0 out the door was finished, some observations about ooDialog were made. One was that previously no real versioning of ooDialog was done. The second was that each of the ooDialogs shipped with ooRexx 4.0.0, 4.0.1, and 4.1.0 only required an ooRexx 4.0.0 interpreter to work, while the refactored ooDialog in the internal development source tree required an ooRexx 4.1.0 interpreter. The third observation was that, since ooDialog is an extension to the Rexx interpreter, there is no reason why it could not be distributed independently of the ooRexx interpreter. Finally, it had been observed from some comments made in bugs opened up against ooDialog, that there were a few people who seemed to prefer to remain with an ooDialog that was designed for Windows 95, in essence an ooDialog unchanged from the ooDialog shipped with ooRexx 3.2.0.

This lead to the decision to define a more formal versioning scheme for ooDialog and also to implement a means to distribute ooDialog independently of the ooRexx distribution. The version string of ooDialog had already been defined to be the version string of the ooRexx interpreter ooDialog was built under. A second part of the ooDialog version was to be the ooDialog level. This is similar to the language level of the Rexx interpreter. The initial ooDialog levels are listed in this table:

Table 1.4. ooDialog Levels

Level	Required Interpreter	Description
4.0.0	4.0.0	ooDialog implementation using Rexx classic external routines and the 3.2.0 ooDialog code.
4.1.0	4.0.0	ooDialog as shipped with ooRexx 4.0.0 with all bug fixes through ooRexx 4.1.2.
4.2.0	4.1.0	Refactored ooDialog completely implemented with the C++ native API.

Level 4.0.0

The ooDialog code at level 4.0.0 is frozen. Since the implementing code uses the classic external routines and the code itself is unchanged from the ooDialog shipped with ooRexx 3.2.0, ooDialog programs should behave exactly as they did under the ooDialog shipped with ooRexx 3.2.0. The only difference is that the ooDialog code at the 4.0.0 level can be compiled for a 64-bit ooRexx installation. No fixes, enhancements, or changes will be made to the code.

ooDialog 4.0.0 is available as a separate download from SourceForge for users who would like to replace their installed ooDialog with a 4.0.0 level ooDialog. It requires that the installed ooRexx is 4.0.0 or later. Although the code is frozen, if a user can demonstrate an ooDialog program behaves differently using a 4.0.0 level ooDialog than it did under an actual ooRexx 3.2.0 installation, a fix will be considered.

Level 4.1.0

ooDialog at level 4.1.0 is the ooDialog that shipped with ooRexx 4.0.0, plus all bug fixes and changes up through ooRexx 4.1.2. It is also provided as a separate download on SourceForge and also requires the installed ooRexx be 4.0.0 or later. This code is frozen, no changes will be made to the code. Bugs found in this code will be fixed in upcoming releases of ooDialog.

It allows users that have not upgraded their ooRexx from 4.0.0 or 4.0.1 to pick up the bug fixes in ooDialog included in the ooRexx distributions up through ooRexx 4.1.1. However, its primary purpose is for users who may be unhappy with the future direction of ooDialog. It will provide a way for users to upgrade to future versions of ooRexx and yet retain an unchanged ooDialog.

Level 4.2.0

This is the actively maintained ooDialog level. The level will be incremented in the future as circumstances dictate. It requires an installed ooRexx of 4.1.0 or later. New versions of ooDialog at the 4.2.0 level will be released as they become ready. It is available as a separate download from SourceForge and will also be the ooDialog level shipped with future versions of ooRexx, (versions past ooRexx 4.1.2.)

Note that as of July 2012, all versions of ooDialog from level 4.0.0 on are available as independent installations and are available for download from SourceForge. Any version of ooDialog can be installed independently to an ooRexx installation, provided the ooRexx installation meets the minimum required for the ooDialog version.

1.4.2. ooDialog Release 4.2.0

ooDialog 4.2.0 was released in late August of 2012. It was the first major enhancement of ooDialog in over a decade. This ooDialog has a lot of new and different things in it. In addition to the new dialog classes, new dialog control classes, new methods on existing classes, it includes design changes intended to simplify areas of ooDialog that seem to have been confusing in the past. Enhancements have been made that fix areas of ooDialog that have always been broken. All ooDialog users should take a close look at these changes.

The over-all number of changes, enhancements, and new methods and classes are too numerous to list in minute detail. The following lists the major areas of change.

Unification of Method and Class Names

The Windows operating system provides an extremely large number of APIs to program GUI applications. To take advantage of this, ooDialog needs to provide a large number of classes and methods in those classes. Over time, and probably due to a number of different programmers working on ooDialog, this lead to a large number of similar, but confusing method and class names.

In ooDialog 4.2.0 an effort has been made to unify the naming of methods and classes. To begin with, each dialog control is named the same as it is in the Microsoft *documentation*. An example of this is the edit control. The name of the control is *edit*. Rather than call an edit control an *entry line*, it is called an edit control. In addition, all dialog controls are controls. Adding *control* to the control's name is redundant, and confusing when it is done in some cases and not in other cases. A list-view is no longer called a ListControl, but rather it is called a ListView, a tree-view is no longer called a TreeControl, it is called a TreeView.

Prior to 4.2.0, there were a large number of method names with the word *connect* in them, with no rhyme or reason to the entire method name. To overcome this, a unified naming scheme has been carried out. All methods with the name of a dialog control in them use the same name as Microsoft's documentation. Method names with connect in them use a similar name for all methods that connect similar things. Methods that connect a data attribute will have *data* in the name. Methods that connect an event notification will have *event* in the name, etc..

The old names of these methods are now *deprecated*, but still work. In order to avoid conflicts with old, deprecated method names, certain groups of methods use a new verb in the name. All methods that return an instantiated dialog control object begin with *new* and contain the control name. I.e., *newTreeView*, *newEdit*, or *newTrackBar*. All methods that create a dialog control in the dialog template of a **`UserDialog`** now begin with the word *create* and include the control name when a single control is being created. For example, *createListView*, *createUpDown*, *createMonthCalendar*. Similar naming changes are in methods connecting data attributes, in methods connecting event notifications, etc..

The primary areas that should be looked at for this unification are the *create...* methods of the UserDialog, methods concerned with connecting *data* attributes, methods returning an *instantiated* dialog control object, and methods that *connect* notifications.

Simplification of Requires Statements

Prior to ooDialog 4.2.0 there were three different ooDialog class files: **`oodPlain.cls`**, **`ooDialog.cls`**, and **`oodWin32.cls`**. An ooDialog program would need to *require* (using a **`::requires`** directive) one or more of these class files, depending on what parts of the ooDialog framework were going to be used. It was often confusing to users as to which files needed to be required. Breaking the ooDialog framework into three pieces served no real purpose. In 4.2.0, the three files have been combined into a single file. All ooDialog programs only need to *require* the **`ooDialog.cls`** file.

Simplification of Inherited Classes

Previously certain dialog controls were deemed *advanced* and to use those controls, or even to use some features common to all controls, the programmer needed to inherit the **`AdvancedControls`** class. This was often confusing, and again there was no real reason for it. The *advanced* controls were only advanced in the previous millennium. To simplify this, the need to inherit the **`AdvancedControls`** class has been eliminated. The methods to work with all controls are part of the base *dialog* object. The *AdvancedControls* class is deprecated.

In 4.2.0 the *MessageExtensions* class is also deprecated. Connecting *event* notifications to a method in the Rexx dialog is such a fundamental part of using a dialog that the programmer should not need to do anything special. The methods to connect events are now all methods of the base classes where they are needed. The methods to connect dialog events are part of the base *dialog*, methods to connect menu events are part of the *Menu* class, methods to connect mouse events are part of the *Mouse* class, etc..

ooDialog 4.2.0 no longer makes a distinction between a *plain* dialog and a *dialog*. The *PlainUserDialog* is deprecated and no longer needed.

True Call Backs from the Windows Message Loop

This is probably the most significant change in ooDialog 4.2.0. Prior to 4.2.0, window message *event* notifications were put into a queue to be dispatched at some later point to the connected method in the Rexx dialog. But, the operating system expects a reply to its event messages, and the reply allows the programmer to customize the behavior of the dialog. Because the programmer could not reply in any way to the operating system, ooDialog was severely restricted in what it could support, compared to a normal Windows application.

ooDialog 4.2.0 has been completely rewritten to use the C++ native APIs introduced in ooRexx 4.0.0. A primary benefit of this is that now the ooDialog framework can directly invoke the connected event handling method in the Rexx dialog and the programmer can directly reply to the operating system message. Further discussion on the benefits of this is included in the section on *coding* event handling methods.

Correct Conversion of Dialog Units to Pixels

The ooDialog framework has always provided the *factorX* and *factorY* attributes, whose intent was to convert between *dialog unit*s and *pixel*s. Unfortunately, the two attributes are based on a fallacy and are *inaccurate*. To compound the problem, many ooDialog methods internally use the attributes to convert between dialog units and pixels, *when no conversion is needed.*

To fix this problem new methods, such as *dlgUnit2pixel*, *pixel2dlgUnit*, etc., have been added that accurately convert between pixels and dialog units. Methods that are inaccurate have had, matching, accurate, methods added to the ooDialog framework. The accurate *getRealPos* is added to match the inaccurate *getPos* method. The accurate *columnWidthPX* is added to match the inaccurate *columnWidth*. The accurate *moveTo* is added to match the inaccurate *move* method, and so on.

New Dialog Classes

New dialog classes have been added that implement true Windows property sheets, wizards, and dialogs that function as controls. These classes replace the deprecated *CategoryDialog* and *PropertySheet* classes. The new dialogs are easier to use and easier to understand because the pages of the dialogs are dialog objects that are programmed and behave using the same paradigm as any other dialog.

These new dialog classes are available in ooDialog 4.2.0:

- *ControlDialog*: Control dialogs work well as a dialog within a top-level dialog. In essence, a **`ControlDialog`** functions as a dialog *control* within the top-level dialog.
- *RcControlDialog*: A **`ControlDialog`** based on the **`RcDialog`**.
- *ResControlDialog*: A **`ControlDialog`** based on the **`ResDialog`** dialog.
- *UserControlDialog*: A **`ControlDialog`** based on the **`UserDialog`** dialog.
- *PropertySheetDialog*: The **`PropertySheetDialog`** is an implementation of the Windows property sheet and the Windows wizard. Unlike most ooDialog dialogs, the operating system does most of the management of these dialogs.
- *PropertySheetPage*: **`PropertySheetPage`** dialogs are used as the pages of property sheets and wizards.
- *RcPSPDialog*: A **`PropertySheetPage`** based on the **`RcDialog`** dialog.
- *ResPSPDialog*: A **`PropertySheetPage`** based on the **`ResDialog`** dialog.
- *UserPSPDialog*: A **`PropertySheetPage`** based on the **`UserDialog`** dialog.

New Dialog Control Classes

Three new dialog control classes are available in ooDialog 4.2.0:

- *DateTimePicker*: A control that provides a simple interface with which to exchange date and time information with the user.
- *MonthCalendar*: A control that provides an easy to use interface which allows users to select or enter dates. The control has the appearance of a typical calendar.
- *UpDown*: A control that consists of a pair of arrow buttons that the user can click to increment or decrement a value, such as a scroll position or a number displayed in a companion control.

The new control classes are fully implemented so that all the functionality of the control, up through Windows 7, is available to the ooDialog programmer.

Menu Classes

In ooDialog 4.2.0 menus are elevated to class objects. The menu classes more fully implement the functionality of Windows menus, rather than just a small subset of the functionality as was done in ooDialog prior to 4.2.0. They also make it easier for the programmer to create the menus needed for an application. These menu classes have been added:

- *BinaryMenuBar*: A **`MenuBar`** created from a binary resource, or by the programmer with the easy to use menu class methods.

- *Menu*: The **`Menu`** class implements the large number of methods that are the same for all menus.

- *MenuBar*: A menu bar is often called the top-level menu. It is the bar positioned just below the title bar of an application window or a dialog. The **`MenuBar`** class implements the methods that are common to all menu bars.

- *PopupMenu*: A menu bar contains submenus, and submenus can also contain submenus. Submenus go by various names, including drop down menus, popup menus, context menus, and shortcut menus. The **`PopupMenu`** class is an implementation of submenus, no matter which name it goes by. Popup menus can be inserted into menu bars, or other submenus. They are also used as context menus. The **`PopupMenu`** class implements the functionality of submenus, including the functionality of true Windows context menus.

- *ScriptMenuBar*: A **`MenuBar`** whose menu template is created by parsing a resource script, (usually a .rc file.) Similar to a *RcDialog*.

- *SystemMenu*: The **`SystemMenu`** class represents the Windows *system* menu, which is also known as the *window* menu or the *control* menu.

- *UserMenuBar*: A **`MenuBar`** whose menu template is created by the programmer using the methods defined by the class. Similar to a *UserDialog*.

Other New Classes

There are a number of new utility and helper classes available in ooDialog 4.2.0:

- *ApplicationManager*: Used to set application wide defaults and to perform tasks that effect the entire ooDialog application.

- *DayState*: Represents the state of each day in a month. A helper class for the *MonthCalendar* control.

- *DayStates*: A sequential collection of **`DayState`** objects. A helper class for the *MonthCalendar* control.

- *Mouse*: Provides all the methods to work with and manipulate the mouse and the mouse cursor.

- *OS*: Provides methods for extracting information about the operating system the program is currently executing on.

- *SM*: The attributes of this class reflect the system metrics or configuration settings of the computer the Rexx program is running on.

- *SPI*: The attributes of this class reflect the system-wide parameters of the computer the Rexx program is running on

- *VK*: The **`VK`** class allows the programmer to use symbolic names in a program instead of the numeric value of the virtual key codes. Unlike the **`VirtualKeyCodes`** class in **`winsystm.cls`** the **`VK`** class has constants for every virtual key code. Another advantage of the **`VK`** class is that the programmer does not need to require **`winsystm.cls`** in his programs.

- *Window*: The **`Window`** class allows the invocation of methods common to every window.

New Methods on Existing Classes

Many, many new methods have been added to both the dialog and the dialog control classes. Too many to list individually. The documentation for each class has been expanded to include a complete method table for the class at the start of the class documentation. The reader can glance through the method table to see which methods the class has, which will give her an idea of the new methods available to that class.

New Utility Objects

ooDialog 4.2.0 includes three new utility objects.

- *.application*: The **`.application`** object is an instance of the *ApplicationManager* class that is present in all ooDialog programs. It is used by the programmer to manage application wide settings, behaviour, and constants.

- *.constdir*) object: Part of the *mechanism* allowing the use of *symbolic* IDs in ooDialog programs. The **`.constDir`** is the most efficient way to use symbolic IDs.

- *.systemErrorCode* object: Can be used, at times, to obtain additional information when the invocation of a method generates an operating system error.

Use of Objects as Arguments and Return Values

Prior to ooRexx 4.0.0, all arguments to and returns from external functions had to be strings. Although stems could also be used, working with them was difficult and inflexible within the external function implementation. With the advent of ooRexx 4.0.0, it became very easy to use any ooRexx object as an argument to, or return from, the ooDialog external library implementation. In addition, instance methods can be implemented directly in the external library.

In the ooDialog released with ooRexx 4.0.0 through ooRexx 4.1.1, a very limited use of passing and returning objects was begun. In ooDialog 4.2.0 extensive use of passing objects is present in the new classes and methods.

Raising of Syntax Conditions

Up through ooDialog 3.2.0 there was very little error checking and reporting in the ooDialog framework. Because of this many things appeared to the user to simply not work. In many cases, the programmer was using incorrect arguments to methods, or using the method incorrectly. In addition, detectable errors reported by the operating system were simply ignored.

In ooDialog 4.2.0 syntax conditions are raised when incorrect usage is detected. There is also the new *.systemErrorCode* object which is set when the operating system reports an error. This is of great benefit to the programmer, especially when writing new programs. Incorrect usage, pointed out by syntax conditions raised the first time the programmer tests his code, allow the programmer to quickly fix mistakes she is making in her code.

Completely New Example Programs

There are many completely new example programs in ooDialog 4.2.0. Many of the new programs demonstrate the new features in ooDialog 4.2.0. The intent with the new examples is to have a range of programs from the simple to the more complex, giving help to newcomers and providing examples of how to move on to more complex real-world applications. The ooDialog programmer should be sure to look through the many programs under the **`samples\oodialog`** directory tree.

Updating of Existing Example Programs

Almost every existing ooDialog example program has been updated. Deprecate methods and classes have been removed. An attempt to better comment the examples has been made. Some examples used *internal use only* methods and this has been corrected. In other examples, the use of questionable techniques has been removed. Even experienced ooDialog programmers can probably gain insight from looking through the existing examples they have seen in previous ooDialog releases.

Revision of the Documentation

The ooDialog reference, this book, has been almost completely rewritten. The old reference was often incomplete, misleading, and in many cases simply wrong. A large effort has been made to expand incomplete areas and to correct misleading or wrong information. That effort is not 100% complete and will need to be finished in future releases. However, the documentation is very much improved. If there were areas in the previous documentation not well understood, the ooDialog programmer is encouraged to reread those sections in this document.

Deprecated Classes and Methods

ooDialog 4.2.0 has introduced the concept of *deprecated* methods and classes. All *deprecated* classes and methods are listed in *Appendix A*. The deprecated classes and methods are no longer documented. When writing new code, the programmer should not use anything deprecated. Wherever possible in existing code, deprecated classes and methods should be replaced.

Backwards Compatibility

The ooDialog implementation has been largely rewritten and refactored. Every effort has been made to maintain backwards compatibility. If a case is found where backwards compatibility is broken, it is likely that it was not intentional, and the case should be brought up with the developers. The *Getting Help and Submitting Feedback* section of this book contains a list of numerous venues through which feedback can be submitted, or questions raised.

There may be a very few areas where maintaining backwards compatibility is not consistent with the intent to move ooDialog past Windows 3.1 and on to Windows 7. If such an area is discovered, the ooDialog developers welcome any reasoned discussion concerning how to deal with that area. Ultimately, the individual programmer may have to decide if the benefits of moving ooDialog into the 21st century outweigh any effort to change existing code.

1.4.3. ooDialog Release 4.2.1

ooDialog 4.2.1 was released in January 2013. It is a 4.2.0 level ooDialog, which means it can be installed to any ooRexx installation that is at least version 4.1.0. The following is a brief summary of the changes in ooDialog 4.2.1 from ooDialog 4.2.0.

Bug Fixes in ooDialog 4.2.1:

The following bugs were fixed in this release. The bug fix numbers can be used to look up details concerning the bug on SourceForge

- * #1116 Debug print outs left in ooDialog 4.2.0
- * #1117 Method handler passed wrong argument for TreeView EXPANDING event
- * #1121 ooDialog appcrash running forward class(super) continue
- * #1124 ooDialog - failed to locate .h file message incorrect
- * #1126 ooDialog - args sent to the HELP event handler incorrect
- * #1128 ooDialog UtilityClasses.cls: typo

- * #1138 MessageDialog: Failure in system service
- * #1141 ooDialog - connecting system menu command events can fail
- * #1145 TimedMessage dialog remains on screen after ok() method is invoked
- * #1146 Return code from ListView::deleteColumn() not as documented
- * #1147 SingleSelection class
- * #1149 Potential crash in ooDialog connectKeyPress() method

Feature Requests Added to ooDialog 4.2.1:

The following Request for Feature Ehancements were implemented in the release. Details concerning the enhancements can be looked up on SourceForge using the listed tracker item numbers.

- * #42 OODialog List Control Class (report) format individual lines
- * #130 OODialog List Control Class Activate sort headers
- * #419 Add Tooltip control
- * #420 Would like to have an AddButtonStem method.
- * #478 ooDialog - List-view could use a way to work with a complete row
- * #483 ooDialog setColor and dialog background improvements
- * #484 ooDialog setColor methods should not be restricted to the 19 palette indexes
- * #485 Treeview itemData
- * #486 ooDialog - the TreeView Expanding event should allow a veto
- * #487 ooDialog - It would be nice if the TreeView class had a way to do custom sorting
- * #488 ooDialog - TreeView class needs a itemText method
- * #489 ooDialog - TreeView class should support info tips
- * #490 ooDialog - ListView class should support info tips
- * #493 ooDialog - TreeView begin / end label editing could be improved
- * #494 ooDialog - ListView begin / end label editing could be improved
- * #495 ooDialog TreeView control has missing styles
- * #496 ooDialog TreeView methods to determine area occupied by an item
- * #499 ooDialog connectTreeViewEvent CHANGING veto
- * #501 The ListView class should have an easy way to switch views

New Functionality in ooDialog 4.2.1:

An implementation that allows access to the Custom Draw facility in Microsoft's Common Control library has been added.

Custom Draw allows, for example, the programmer to specify the text and background colors for individual rows in a ListView, even for individual columns in the row. The programmer can specify the font of the text for individual list-view items. Etc., etc..

The CustomDraw class has been added to ooDialog and is the means to access the Custom Draw facilities.

New ListView support classes in ooDialog 4.2.1:

LvItem class: Represents a single list-view item. The attributes of the LvItem are the attributes of the underlying Windows list-view item.

LvSubItem class: Represents a single list-view subitem. The attributes of the LvSubItem are the attributes of the underlying Windows list-view subitem.

LvFullRow class: List-view items can contain subitems. This is most apparent in the row view of a list-view where the subitems are displayed in columns next to the item. However, the subitems, once added to an item, are always present, even if the list-view is in another view, such as icon view.

The LvFullRow class represents the list-view item and all its subitems. LvFullRow objects can be used to insert an item and all data associated with the item into a list-view at one time.

LvCustomDrawSimple class: The LvCustomDrawSimple class is used in conjunction with the custom draw facility for ListView controls.

New dialog classes in ooDialog 4.2.1:

CustomDraw class: The CustomDraw class is a new mixin class. Dialogs in ooDialog can inherit this class, which then gives the dialog the ability to access the custom draw facility.

The ListView and TreeView controls can register for custom draw.

New dialog control classes in ooDialog 4.2.1:

ToolTip class: Tooltip controls are pop-up windows that display text. Typically the text describes a *tool*. A *tool* is either a window or an application defined area within a window.

New utility classes in ooDialog 4.2.1:

TvCustomDrawSimple class: The TvCustomDrawSimple class is used in conjunction with the custom draw facility for TreeView controls.

New Methods in ooDialog 4.2.1:

In the PlainBaseDialog class:

getTextSizeTitleBar

In the DialogControl class:

useVersion.

usingVersion.

In the ListView class:

addFullRow

insertFullRow

prependFullRow

In the TreeView class:

find

getItemData

itemText

removeItemData

setItemData

New Attributes in ooDialog 4.2.1:

In the SM (system metrics) class:

cxSize

cxSmIcon.

cyMenu.

Enhanced Methods in ooDialog 4.2.1:

In the DialogControl class:

setColor

setSysColor

In the DialogExtensions class:

setControlColor.

setControlSysColor.

In the TreeView class:

insert

itemInfo

modify

In the EventNotification class:

connectListViewEvent

- the KEYDOWNEX event is added.
- the BEGINEDIT event is enhanced
- the ENDEDIT event is enhanced

connectTreeViewEvent

- the KEYDOWNEX event is added.
- the BEGINEDIT event is enhanced
- the ENDEDIT event is enhanced

New samples in ooDialog 4.2.1:

`oodialog\controls\ListView\customDrawListView.rex`: The customDrawListView example demonstrates how to change the text, text color, and background color of individual rows in a list-view.

`oodialog\controls\ListView\columnIcons.rex`: The columnIcons.rex example demonstrates how to use the new LvFullRow class and how to use icons in the columns of list-view items in report view.

`oodialog\controls\ToolTip\toolTip.rex`: The toolTip.rex example demonstrates the basics of using ToolTips in ooDialog. It includes the usage of a number of the ToolTip methods.

`oodialog\controls\ToolTip\customPositionToolTip.rex`: The customPositionToolTip.rex example shows how to use the ToolTip class to do custom positioning of the info tips provided by the tree-view.

`oodialog\controls\ToolTip\manageControlTool.rex`: The manageControlTool.rex example shows how to use some advanced features of the ToolTip class to provide completely customized ToolTips for a dialog control. This example uses a tree-view control, but the techniques could be applied to any dialog control.

Enhanced samples in ooDialog 4.2.1:

`oodialog\controls\TreeView\treeViewCustomDraw.rex`: The oodtree.rex example has been renamed to the treeViewCustomDraw.rex and enhanced. The example has been more fully commented, uses custom draw for the tree-view control, and has a number of small bugs fixed.

Documentation changes in ooDialog 4.2.1:

The ListView Control chapter in the ooDialog Reference Manual has been partially reviewed and corrected for accuracy. Parts of the chapter have been updated to match the format and style that was introduced in the rework of the manual done for ooDialog 4.2.0

The TreeView Control chapter in the ooDialog Reference Manual has been partially reviewed and corrected for accuracy. Parts of the chapter have been updated to match the format and style that was introduced in the rework of the manual done for ooDialog 4.2.0

The documentation for the connectListViewEvent and connectTreeViewEvent methods have been enhanced.

1.4.4. ooDialog Release 4.2.2

ooDialog 4.2.2 was released in March 2013. It is a 4.2.0 level ooDialog, which means it can be installed to any ooRexx installation that is at least version 4.1.0. The following is a brief summary of the changes in ooDialog 4.2.2 from 4.2.1.

Bug Fixes in ooDialog 4.2.2:

The following bugs were fixed in this release. The bug fix numbers can be used to look up details concerning the bug on SourceForge

- * #1153 Deprecated CategoryDialog dialogs do not work in ooDialog 4.2.1
- * #1154 setColor() method produces incorrect results
- * #1166 Naming a DlgAreaU object 'b' causes the object to fail

Documentation Bug Fixes in ooDialog 4.2.2:

The following documentation bugs were fixed in this release. The bug fix numbers can be used to look up details concerning the bug on SourceForge

- * #196 ooDialog - mapWindowPoints() was never documented

Feature Requests Added to ooDialog 4.2.2:

The following Request for Feature Ehancements were implemented in the release. Details concerning the enhancements can be looked up on SourceForge using the listed tracker item numbers.

- * #249 Add support for generalized resizable dialogs

- * #352 Fix document StateIndicator / ProgressIndicator
- * #370 Replace ooDialog WinTimer with a waitable time
- * #417 ooDialog public routines and icon resource
- * #504 It would be nice to be able to store a user value as part of the LvFullRow object
- * #505 Would like a way to prevent Enter from closing dialog when in single-line edit control
- * #506 Allow modifying a list-view item and all its subitems at once
- * #507 ooDialog - Modify the text of a list-view item and all its subitems at one time
- * #512 Add setWindowStyleRaw to match WindowBase getWindowStyleRaw
- * #513 Need access to the ListView_GetItemRect ListView_GetIndexItemRect
- * #514 It would be interesting to temporarily change the parent window of a dialog control
- * #515 Would like to connect list-view begin and end scroll events
- * #517 Add a way to access the child controls of a combo box
- * #518 Need to work with an edit control that is not a direct child of the dialog
- * #519 Would like the ListView hitTestInfo to return the subitem index
- * #520 folder selection dialog
- * #521 It would be useful to create a LvFullRow from an array of text values
- * #522 Add item to list-view using array to specify the text of the item / subitem(s)
- * #524 Standard dialogs should work no matter what the application wide setting of auto detect
- * #525 The TreeView find() should allow some options
- * #526 Get the message text associated with Windows error codes
- * #528 Get the list box item under a point
- * #529 Add methods to determine type of combo box from Rexx code
- * #530 Want to be be able to calculate the rectangle occupied by a combo box item
- * #531 Combo box methods to set the minimum number of items shown in list
- * #532 Set the cue banner text for a combo box
- * #534 Key status
- * #535 Treeview expand/collapse button replacement
- * #536 popupAsChild support for PropertySheets

New Funtionality in ooDialog 4.2.2:

Methods have been added to various classes that allow the embedding of either an edit control or a combo box control in the subitem fields of a list-view. This functionality allows the in-place editing of the subitems in a list-view item.

Work has been done that allows any type of ooDialog dialog to be made resizable.

New Dialog Classes in ooDialog 4.2.2:

ResizingAdmin class: The ResizingAdmin class is a new mixin class. Dialogs that inherit this class are automatically made resizable. Any ooDialog dialog class can inherit the mixin class. By default, all controls in the dialog are sized in proportion to the change in size of the dialog. Methods of the ResizingAdmin class allow the programmer to customize how the controls in the dialog are sized.

New Utility Classes in ooDialog 4.2.2:

BrowseForFolder class: The ooDialog BrowseForFolder class provides access to the Windows BrowseForFolder COM object. It gives the Rexx programmer the ability to put up a standard dialog that allows the user to select a folder. It is somewhat similar the fileNameDialog() routine, except it is possible to select a folder. Something that has not been possible in ooDialog previously. The programmer has access to the full set of customizations in the COM object.

SimpleFolderBrowse class: The SimpleFolderBrowse class also provides access to the Windows BrowseForFolder COM object, however it is designed to be very simple to use. Only the most common customization features are provided.

Keyboard class: The Keyboard class addresses the receiving and processing of keyboard input.

ProgressDialog class: A ProgressDialog object is used to show a dialog containing a ProgressBar, an optional message, an optional status line, and an optional Cancel button. The dialog shows the progress a lengthy task is making to the user.

New Public Routines in ooDialog 4.2.2:

SimpleFolderBrowse routine: This routine provides a shortcut access to the SimpleFolderBrowse dialog class.

New Methods in ooDialog 4.2.2:

In the ApplicationManager class:

defaultIcon

In the ComboBox class:

getCue

getCombBoxInfo

getEditControl

getFirstVisible

getItemHeight

getMinVisible

isDropDown

isDropDownList

isGrandchild

isSimple

setCue

setItemHeight

setMinVisible

In the DialogControl class:

setParent

setWindowTheme

In the DlgUtil class:

errMsg

In the Edit class:

isGrandchild

wantReturn

In the ListBox class:

hitTestInfo

In the ListView class:

addRowFromArray.

getItemRect.

getSubitemRect.

modifyFullRow.

setFullRowText.

In the LvFullRow class:

fromArray

In the PropertySheetDialog class:

popupAsChild

In the Rect class:

copy

In the WindowBase class:

setStyleRaw

New Attributes in ooDialog 4.2.2:

In the LvFullRow class:

userData

In the PlainBaseDialog class:

dlgID

Enhanced Methods in ooDialog 4.2.2:

In the Button class:

getIdealSize: A new optional parameter, wantWidth, is added. If this argument is specified, the operating system will calculate the ideal height of the button with the specified width. This functionality is not available in the operating system until Vista or later.

In the EventNotification class:

connectListViewEvent: New events added.

- The BEGINSCROLL event is added.
- The ENDSCROLL event is added.

In the ListView class:

hitTestInfo: Enhanced to return the subitem index, instead of 0.

In the TreeView class:

find: Enhanced to allow starting the search from any parent tree-view item. Enhanced to allow searching for an abbreviation of the tree-view item label.

Enhanced Public Routines in ooDialog 4.2.2:

WinTimer: The WinTimer() public routine was originally implemented in a way that caused it consume 100% of the CPU in a busy loop. The routine has been re-implemented in a way that keeps the behaviour of the routine the same, without the high CPU use. The CPU use while waiting on the timer is now 0%.

Deprecated Methods in ooDialog 4.2.2:

In the Treeview class:

hitTest: This method is deprecated. Use the*hitTestInfo* method instead.

New Samples in ooDialog 4.2.2:

These 3 examples demonstrate how to embed a control in a list-view when it is in report view and use the control to do in-place subitem editing.

- **`ooDialog\controls\ListView\subitem.editing\dropDownComboBox.rex`**
- **`ooDialog\controls\ListView\subitem.editing\dropDownListComboBox.rex`**
- **`ooDialog\controls\ListView\subitem.editing\editControl.rex`**

These 3 examples demonstrate inheriting the ResizingAdmin class.

- **`oodialog\resizableDialogs\ResizingAdmin\augmentedResize.rex`**
- **`oodialog\resizableDialogs\ResizingAdmin\basicResize.rex`**
- **`oodialog\resizableDialogs\ResizingAdmin\gbStationary.rex`**

This example shows how to use the new ProgressDialog class.

- **`oodialog\examples\addManyRows.rex`**

This example demonstrates the BrowseForFolder class.

- **`oodialog\examples\browsePrinters.rex`**

This example demonstrates the SimpleBrowseForFolder class.

- **`oodialog\examples\simpleFolderBrowse.rex`**

This example demonstrates general ooDialog programming techniques by putting up a *stop watch* dialog.

- **`oodialog\examples\stopWatch.rex`**

Enhanced Samples in ooDialog 4.2.2:

This example is an enhanced version of the old oostddlg.rex. It has a completely graphical interface and no longer uses 'say' statements. It also uses the defaultIcon() method to set the default icon for the program.

- **`oodialog\oodStandardDialogs.rex`**

This example is an enhanced version of the old oostdfct.rex. It has a completely graphical interface and no longer uses 'say' statements. It also uses the defaultIcon() method to set the default icon for the program.

- **`oodialog\oodStandardRoutines.rex`**

These 2 examples are copied from the **`oodialog\propertySheet.tabControls\PropertySheetDemo.rex`** and **`oodialog\propertySheet.tabControls\TabDemo.rex`** examples and enhanced to be resizable. The old and new examples can be compared to see both how to make a dialog resizable and how the appearance of the old and new versions contrast.

- **`oodialog\resizableDialogs\ResizingAdmin\PropertySheetDemo.rex`**
- **`oodialog\resizableDialogs\ResizingAdmin\TabDemo.rex`**

Documentation Changes in ooDialog 4.2.2:

All new methods, new classes, new functionality, etc., are fully documented. Other than that there are no major changes to the documentation.

1.4.5. ooDialog Release 4.2.3

ooDialog 4.2.3 was released in ??? YYYY. It is a 4.2.0 level ooDialog, which means it can be installed to any ooRexx installation that is at least version 4.1.0. The following is a brief summary of the changes in ooDialog 4.2.3 from 4.2.2.

Bug Fixes in ooDialog 4.2.3:

The following bugs were fixed in this release. The bug fix numbers can be used to look up details concerning the bug on SourceForge

- * #1175 Incorrect super class initialization can bypass DynamicDialog init
- * #1176 ooDialog - the SingleSelection class has the potential of not executing
- * #1198 ooDialog installer aborts if oodialog.pdf does not exist
- * #1204 When parsing a .rc file ooDialog does not recognize some control statements

Documentation Bug Fixes in ooDialog 4.2.3:

The following documentation bugs were fixed in this release. The bug fix numbers can be used to look up details concerning the bug on SourceForge

- * #201 getTextAlign(), getTextExtent(), setTextAlign() not documented
- * #202 ooDialog Reference 4.2.3 - Text errors in Section 8.1 Method Table

Feature Requests Added to ooDialog 4.2.3:

The following Request for Feature Ehancements were implemented in the release. Details concerning the enhancements can be looked up on SourceForge using the listed tracker item numbers.

- * #508 ooDialog - index for tree-view state images

- * #537 Add FIle Open Dialog to allow more options than available with the FileNameDialog
- * #540 Add a stand alone ooDialog program launcher
- * #549 The setColor() method does not work well with comboboxes
- * #554 Enhance stand alone ooDialog installer to use components
- * #555 .OpenFileDialog set the 'CANCEL' button label
- * #556 Would like access to the Win32 UpdateWindow() API
- * #559 ooDialog - add a convenience function to help locate resource files

Miscellaneous Fixes in ooDialog 4.2.3:

Fixed a problem detecting some controls in the .rc files produced by some resource editors. Combo box, edit, list box, and scroll bar.

New Funtionality in ooDialog 4.2.3:

The stand alone ooDialog installer now allows the user to choose to skip the installation of the documentation and / or the example programs. This matches the ooRexx installation where the documentation and sample programs do not have to be installed.

An ooDialog executable, *ooDialog.exe*, has been added to the distribution. The executable will run ooDialog programs in a manner similar to rexxhide, provides command line information on the version of ooDialog installed, and can be used to set up file types and associations for ooDialog programs.

New Dialog Classes in ooDialog 4.2.3:

CommonItemDialog

OpenFileDialog

SaveFileDialog

New Utility Classes in ooDialog 4.2.3:

ComConstants

CommonDialogEvents

CommonDialogCustomizations

ShellItemFilter

New Public Routines in ooDialog 4.2.3:

Locate

New Methods in ooDialog 4.2.3:

In the **`ApplicationManager`** class:

requiredOS

In the **`ComboBox`** class:

removeFullColor

setFullColor

In the **`DlgUtil`** class:

getGUID

In the **`TreeView`** class:

getStateImage

setStateImage

In the **`WindowBase`** class:

updateWindow

New Attributes in ooDialog 4.2.3:

In the **`ApplicationManager`** class:

srcDir

Enhanced Methods in ooDialog 4.2.3:

In the **`TreeView`** class:

getImageList

setImageList

Enhanced Public Routines in ooDialog 4.2.3:

None.

Deprecated Methods in ooDialog 4.2.3:

None.

New Samples in ooDialog 4.2.3:

Demonstrates what the 3 types of combo boxes look like. Shows how to use the removeFullColor() and setFullColor() methods of the ComboBox class.

- **`oodialog\controls\ComboBox\comboBoxTypes.rex`**

Shows how to implement an incremental search feature for the items in a list box and over-ride the list box control's internal incremental search.

- **`oodialog\controls\ListBox\incrementalSearch.rex`**

An example showing how to use the .DlgUtil's genGUID() method. Also provides a simple application that can be used to generate GUIDs for any use.

- **`oodialog\examples\genGUID.rex`**

An example showing a number of different uses of both the Open File Dialog and the Save File Dialog. Demonstrates the most common usage patterns.

- **`oodialog\examples\openSaveFileDemo.rex`**

An example showing how to use the Common Item Dialog in file save mode. This example shows 2 things. 1.) How to use a filter with the dialog. This is done with the .ShellItemFilter class. 2.) How to connect event notifications. This is done with the .CommonDialogEvents class.

- **`oodialog\examples\saveFileWithFilter.rex`**

An example showing how to embed controls in a list view to allow in-place subitem editing. The example also demonstrates the ResizingAdmin and BrowseForFolder classes.

- **`oodialog\controls\ListView\subitem.editing\importList.rex`**

Enhanced Samples in ooDialog 4.2.3:

All of the samples have been enhanced so that they can locate any needed resource files no matter which directory they are executed from. This allows them to run correctly if, for example, they are dragged and dropped on ooDialog.exe.

Documentation Changes in ooDialog 4.2.3:

All changes in ooDialog 4.2.3 have been fully documented. Other than that, only small progress has been made in reviewing older sections of the reference manual for accuracy.

1.5. Current Release

The current release of ooDialog is 4.2.4. It is a 4.2.0 level ooDialog, which means it can be installed to any ooRexx installation that is at least version 4.1.0. The following is a brief summary of the changes in ooDialog 4.2.4 from 4.2.3.

Bug Fixes in ooDialog 4.2.4:

The following bugs were fixed in this release. The bug fix numbers can be used to look up details concerning the bug on SourceForge

- *
- *

Documentation Bug Fixes in ooDialog 4.2.4:

The following documentation bugs were fixed in this release. The bug fix numbers can be used to look up details concerning the bug on SourceForge

- *

Feature Requests Added to ooDialog 4.2.4:

The following Request for Feature Ehancements were implemented in the release. Details concerning the enhancements can be looked up on SourceForge using the listed tracker item numbers.

- *
- *

New Funtionality in ooDialog 4.2.4:

xx

xx

New Dialog Classes in ooDialog 4.2.4:

xx

New Utility Classes in ooDialog 4.2.4:

xx

New Public Routines in ooDialog 4.2.4:

xx

New Methods in ooDialog 4.2.4:

In the XX class:

xx

In the YY class:

yy

New Attributes in ooDialog 4.2.4:

In the XX class:

xx

Enhanced Methods in ooDialog 4.2.4:

In the XX class:

xx

Enhanced Public Routines in ooDialog 4.2.4:

xx

Deprecated Methods in ooDialog 4.2.4:

In the ZZ class:

xx

New Samples in ooDialog 4.2.4:

xxx

- **`ooDialog\xxx.rex`**

Enhanced Samples in ooDialog 4.2.4:

xxx

- **`oodialog\xxx.rex`**

Documentation Changes in ooDialog 4.2.4:

xxx

1.6. Future

The current future direction of ooDialog is to update ooDialog to take full advantage of the Windows 7 functionality for graphical interfaces, related to dialogs. This includes adding more of the missing dialog controls, such as tool bars, status bars, ip address controls, etc.. It also includes updating existing dialog controls to the full functionality available to them in Windows 7.

There are a number of new classes introduced in ooDialog 4.0.0 through 4.2.0 that are intended to be enhanced in future releases. The *Image* and *ImageList* classes to name just a few. Work will continue on extending those types of classes. In addition work will be done on implementing features requested by ooDialog programmers.

Work will continue on improving the ooDialog reference manual. Sections that were not reviewed for this release will be reviewed and corrected. It is intended that the ooDialog User Guide will continue to be improved.

The users of ooDialog can take part in directing the future of ooDialog through discussion on the Users Mailing List and by opening requests for enhancements at the SourceForge ooRexx project. Users can subscribe to the oorexx-users mailing list on the *ooRexx Mailing List Subscriptions*[1] page. Requests for ooDialog features can be made on the *ooRexx Feature Requests*[2] page.

[1] http://sourceforge.net/mail/?group_id=119701

1.7. ooDialog Class Reference

The classes provided by ooDialog form a hierarchy as shown in The *Hierarchy* of ooDialog Classes. ***Unfortunately*** this section is very out of date. It will be updated in a future version of the ooDialog reference.

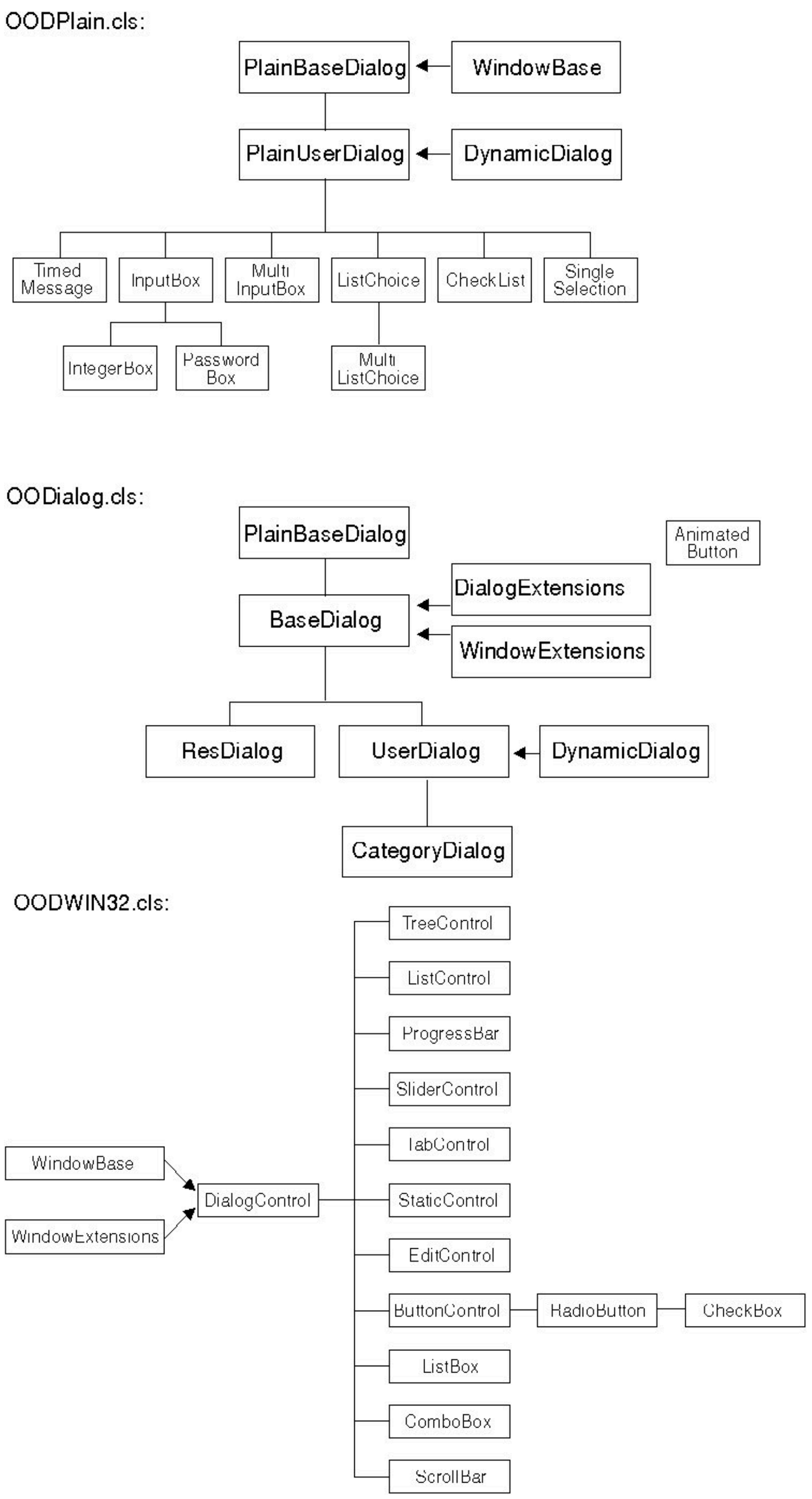


Figure 1.1. The Hierarchy of ooDialog Classes

[2] http://sourceforge.net/tracker/?group_id=119701&atid=684733

The classes are:

PlainBaseDialog, BaseDialog
: Base methods regardless of whether the dialog is implemented as a binary resource, a script, or dynamically. PlainBaseDialog provides limited functionality.

DynamicDialog, DialogExtensions, WindowBase, WindowExtensions
: Internal mixin classes used to extend PlainBaseDialog, PlainUserDialog, BaseDialog, UserDialog, and DialogControl. The methods provided by these classes are not listed separately but are listed in BaseDialog or UserDialog.

UserDialog
: Subclass of BaseDialog used to create a dialog with all its control elements, such as push buttons, check boxes, radio buttons, entry lines, and list boxes.

ResDialog
: Subclass of BaseDialog for dialogs within a binary (compiled) resource file (.DLL).

TimedMessage
: Class to show a message window for a defined duration.

InputBox
: Class to dynamically define a dialog with a message, one entry line, and two push buttons (OK, Cancel).

PasswordBox
: Similar to InputBox, but keystrokes in the entry line are shown as asterisks (*).

IntegerBox
: Similar to InputBox, but only numeric data can be entered in the entry line.

MultiInputBox
: Similar to InputBox, but with multiple entry lines.

ListChoice
: Class to dynamically define a dialog with a list box, where one line can be selected and returned to the caller.

MultiListChoice
: Similar to ListChoice, but more than one line can be selected and returned to the caller.

CheckList
: Class to dynamically define a dialog with a group of check boxes, which can be selected and returned to the caller.

SingleSelection
: Class to dynamically define a dialog with a group of radio buttons, where one can be selected and returned.

Dialog
: Subclass of UserDialog for simple dialogs. You can change the default dialog style from UserDialog to ResDialog.

AnimatedButton
: Class to implement an animated button within a dialog.

DialogControl
: Class to implement methods that are common to all dialogs and dialog controls.

TreeView
: Class to implement a tree to display the list of items in a dialog in a hierarchy.

ListView
: Class to implement a list-view to display the items in a dialog as a collection.

ProgressBar
: Class to implement a progress indicator within a dialog.

Trackbar
: Class to implement a slider or trackbar within a dialog.

Tab
: Class to implement tabs, which can be compared to dividers in a notebook or labels in a file cabinet.

Static
: Class to query and modify static controls, such as static text, group boxes, and frames.

Edit
: Class to query and modify edit controls, which are also called entry lines.

Button
: Class to implement push buttons within a dialog.

RadioButton
: Class to implement radio buttons within a dialog.

CheckBox
: Class to implement check boxes within a dialog.

ListBox
: Class to implement list boxes within a dialog.

ComboBox
: Class to implement a combo box, which combines a list box with an edit control.

ScrollBar
: Class to implement a scroll bar within a dialog.

The ooDialog Executable

The distribution of ooDialog includes an executable file, named: **`ooDialog.exe`**. This executable file is located in the same directory as the **`oodialog.dll`** and **`ooDialog.cls`** files. In the typical installation of ooRexx or ooDialog, this directory will be in the PATH which allows ooDialog.exe to be accessed from anywhere on the computer.

2.1. OverView

The ooDialog executable provides a number of enhancements and services to use along with the ooDialog framework. These range from the simplistic to the relatively complex. There is no need, or requirement, to ever run the executable if the ooDialog user is not interested in any of the features provided by **`ooDialog.exe`**. That is, currently, there are no dependencies on **`ooDialog.exe`** in the ooDialog framework.

The following briefly lists and describes the functionality provided by the ooDialog executable:

Displays information about the installed ooDialog:
: ooDialog.exe can be used to display simple version information for the installed ooDialog framework. It also will display more detailed information for the ooDialog installation such as the build date, 32 or 64 bit, debug or release, version of ooRexx that is in use, etc..

Executes ooRexx programs without a console window:
: ooDialog.exe can be used as a replacement for rexxhide.exe to execute ooRexx programs without a secondary console window popping up. ooDialog.exe has a few enhancements that may make it more suitable to use than rexxhide.exe.

Serves as a drop target for drag and dropping ooRexx programs:
: ooRexx programs can be dragged and dropped on ooDialog.exe, or a short cut to ooDialog.exe, to execute them.

Configures file associations for ooDialog and the PATHEXT
: For users not familiar with setting up file types and associations, ooDialog.exe can present a graphical user interface to configure file types and associate them with the executable.

2.2. ooDialog.Exe Command Line Options

One of the simplest features ooDialog.exe provides is to display information about the current installation of ooDialog. Since the installation of ooDialog has been decoupled from the installation of ooRexx, the version of ooRexx no longer has any relation to the installed version of ooDialog. ooDialog.exe can be executed from a command prompt, using the **`-v`** option, to quickly determine the version of the installed ooDialog:

```
C:\Rexx>oodialog -v
ooDialog 4.2.4.9574

C:\Rexx>
```

If ooDialog.exe is executed from a command prompt with no options, or if it is double-clicked in Explorer, it opens an informational window that displays the ooDialog version and other details concerning the executable and its purpose. Executing ooDialog.exe from a command prompt with the **`-h`** option displays the syntax for ooDialog.exe. For example:

```
C:\Rexx>oodialog -h
Syntax:

    ooDialog [option flags] [programName] [arg1 arg2 ... argN]

option flags:
  -h        Show this, the short help text.
  /?        Same as -h.
  -H        Show the long help text.
  --help    If from a console window, show the long help.
  -s        Display the ooDialog Setup dialog.
  -v        Print the short version string.
  -V        Print the long version string.
  --version If from a console window, print the long version string.

programName:
  The first argument that does not begin with '-' is taken as the name
  of a Rexx program to be executed.

arg1 ... argN:
  Arguments 1 through N are passed on to programName.


C:\Rexx>
```

Executing ooDialog.exe from a command prompt with the **`--version`** option displays more detailed information concerning the ooDialog version, along with the version of ooRexx that ooDialog is running under. For example:

```
C:\Rexx>oodialog --version

ooDialog: ooDialog Version 4.2.4.9574 (64 bit) - Internal Test Version
          Built Nov 27 2013 07:38:21
          Copyright (c) IBM Corporation 1995, 2004.
          Copyright (c) RexxLA 2005-2013.
          All Rights Reserved.

Rexx:     Open Object Rexx Version 4.2.0


C:\Rexx>
```

2.3. ooDialog.Exe as a Program Launcher

A primary use of ooDialog.exe is to execute ooDialog programs without displaying a secondary console window. This is very much like the rexxhide.exe program that comes with the Windows ooRexx distribution. The ooRexx interpreter (rexx.exe) is built as a console application. This is what gives the interpreter access to the standard input and output streams. Which in turn allows the *say* instruction to write text to the console and the *pull* instruction to retrieve text the user has typed.

Because it is a console application, whenever rexx.exe starts, the operating system creates a console window if it is not already being executed from a console window. For a purely graphical application, as most ooDialog programs are, this secondary console window can be an annoyance. The ooDialog executable, (and rexxhide.exe,) are built as a windowed application and internally use the ooRexx naive API to start an instance of the interpreter to execute the Rexx program. Because they are built as a windowed application, no console window is created by the operating system.

One drawback that rexxhide.exe has, is that since it is built as a windowed application with no console window, all text normally written to the console is lost. This includes the error condition text written when a condition is raised. Because of this Rexx programs run under rexxhide.exe just silently end if a problem in the program raises a condition. This can be frustrating because the program will just terminate and there is no way to know why.

The ooDialog executable has been enhanced to help with this situation. If a condition is raised and the Rex program is terminated, ooDialog.exe traps the condition text and then displays it in a window when the interpreter terminates.

Note

This discussion on using ooDialog.exe as a program launcher mostly refers to launching ooDialog programs. However, ooDialog.exe will execute any ooRexx program. It is not restricted to launching only ooDialog programs.

There are a number of ways to launch ooDialog programs using ooDialog.exe:

direct invocation:

From a command prompt simply use ooDialog.exe instead of rexx.exe:

```
/* Instead of this: */

C:\Rexx>rexx.exe ooRexx\ooRexxTry.rex

/* Use this: */

C:\Rexx>ooDialog.exe ooRexx\ooRexxTry.rex
```

within a shortcut or Start menu item:

Typically, people create a short cut to an ooDialog program using rexxhide.exe. Then, double-clicking the short cut in Explorer launches the ooDialog program with no console Window. The ooRexx installer has set up these types of short cuts for ooDialog example programs for years. In the short cut, ooDialog.exe can be used instead of rexxhide.exe:

```
/* The ooRexx installer has set up a Start menu item to the ooDialog sample.rex program
 like this: */

Target type:      Application
Target location:  ooRexx
Target:           C:\Rexx\ooRexx\rexxhide.exe "C:\Rexx\ooRexx\samples\oodialog
\sample.rex"
Start in:         C:\Rexx\ooRexx\samples\oodialog\

/* Use this for the short cut instead: */

Target type:      Application
Target location:  ooRexx
Target:           C:\Rexx\ooRexx\ooDialog.exe "C:\Rexx\ooRexx\samples\oodialog
\sample.rex"
Start in:         C:\Rexx\ooRexx\samples\oodialog\
```

with drag and drop:

The ooDialog executable is drag and drop enabled. That is, ooDialog.exe can serve as a drop target. In Explorer when an ooDialog program file is dragged and dropped on the ooDialog.exe icon, the ooDialog program will execute without a secondary console window. A common use of this ability would be to create a short cut to ooDialog.exe on the desktop and drag and drop ooDialog programs on the short cut.

This snapshot of a portion of a user's Desktop shows a short cut icon to ooDialog.exe. ooDialog programs such as **`columnIcons.rex`** can be dragged from Explorer and dropped on the short cut, which launches the ooDialog program.

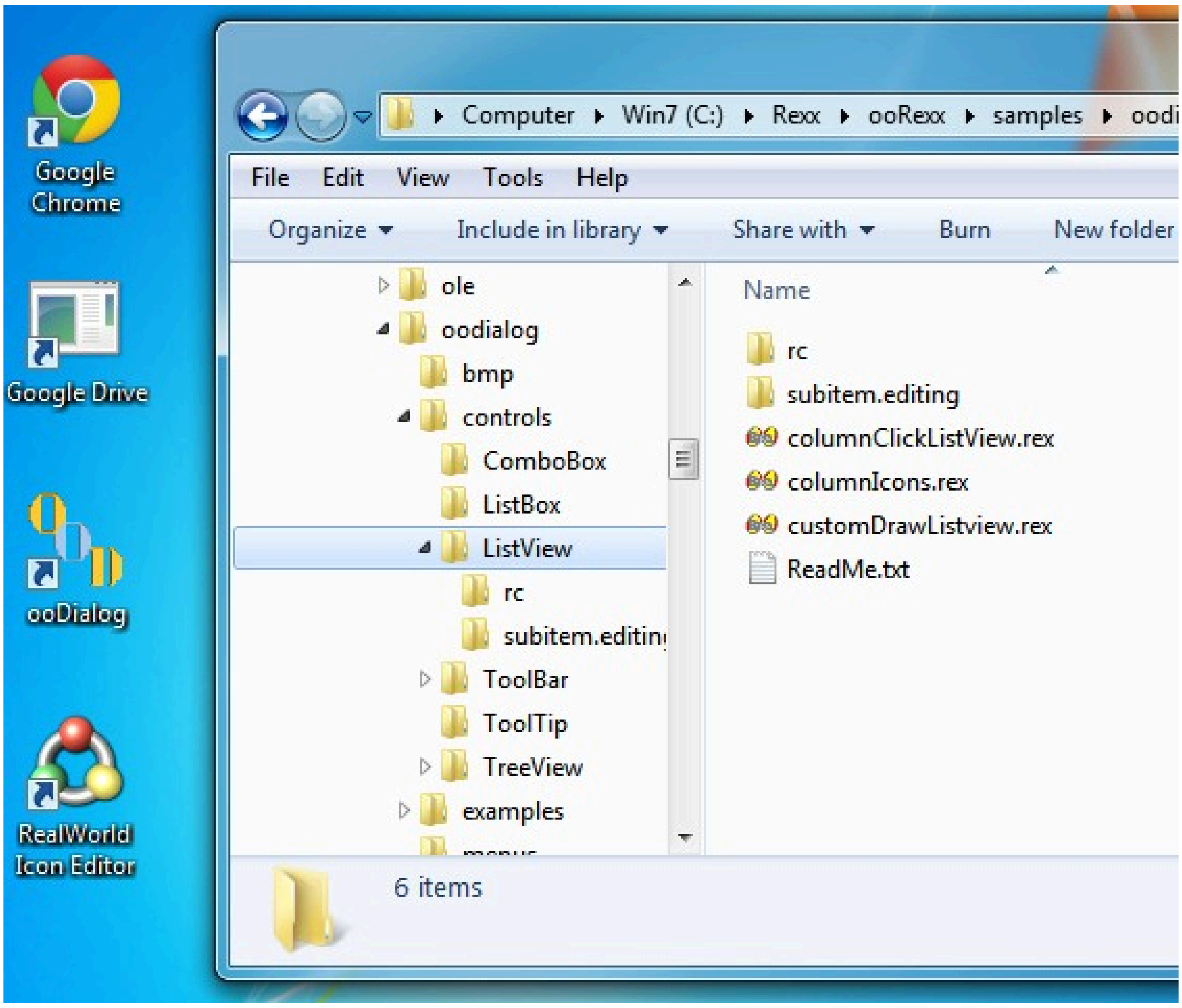
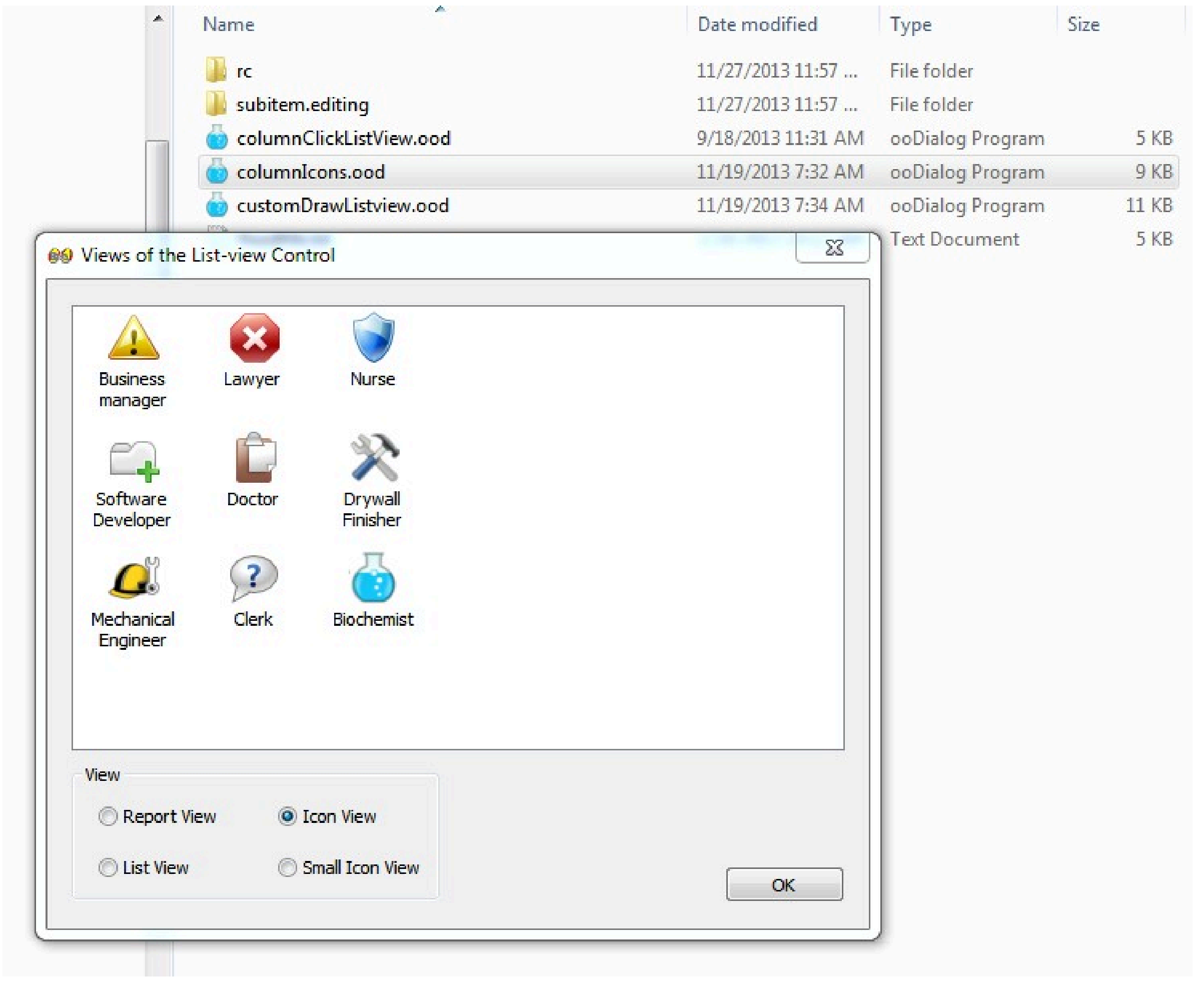


Figure 2.1. Drag and Drop Scenario

using file types and associations:

Probably one of the best ways to use ooDialog.exe to execute ooDialog programs is through the use of file type associations. Once a file type has been created for ooDialog.exe, and an extension associated with that file type, double-clicking on any file with the extension will execute the ooDialog program.

The figure below shows a user's system where a file type has been created for ooDialog.exe and the extension *.ood* is associated with that file type. Double-clicking on **`columnIcons.ood`** has executed that ooDialog program.



Name	Date modified	Type	Size
rc	11/27/2013 11:57 ...	File folder	
subitem.editing	11/27/2013 11:57 ...	File folder	
columnClickListView.ood	9/18/2013 11:31 AM	ooDialog Program	5 KB
columnIcons.ood	11/19/2013 7:32 AM	ooDialog Program	9 KB
customDrawListview.ood	11/19/2013 7:34 AM	ooDialog Program	11 KB
		Text Document	5 KB

View: Report View [✓ Icon View] List View Small Icon View

Figure 2.2. .ood Extension Associated with ooDialog.exe

Although it is not discernible from the figure itself, the created file type on the user's system is **`ooRexx.ooDialog.1`**. File types and associations are relatively easy to create and manage from the command prompt, or through Explorer. People familar will file type associations will recognize that the following displays the file type and association from the system in the figure above:

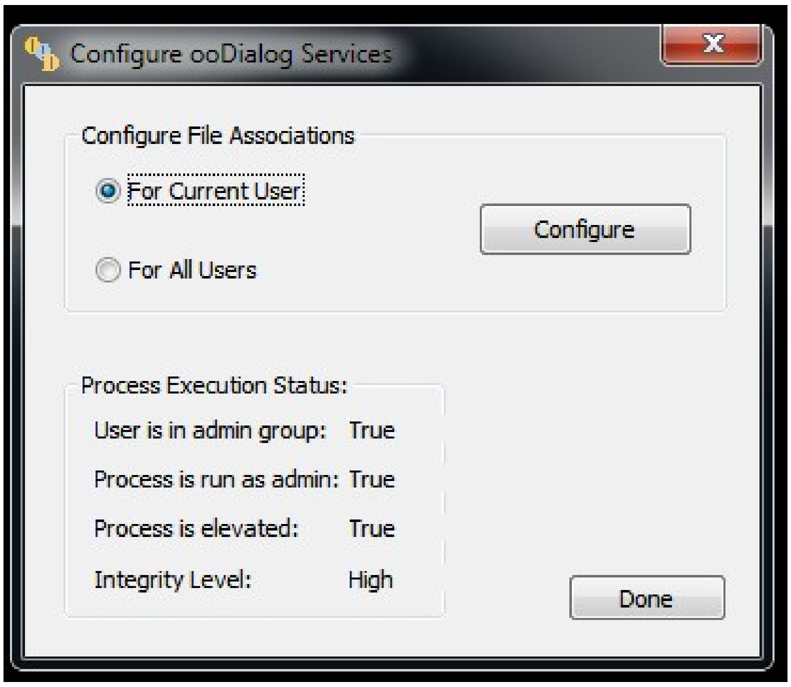
```
C:\Rexx>ftype ooRexx.ooDialog.1
ooRexx.ooDialog.1="C:\Rexx\ooRexx\ooDialog.exe" "%1" %*

C:\Rexx>assoc .ood
.ood=ooRexx.ooDialog.1

C:\Rexx>
```

For people not familar with creating file types and associations, ooDialog.exe can be used to configure file associations for the system through a graphical user interface. This use of ooDialog.exe is explained in the next section

2.4. ooDialog.Exe Configure File Associations

Windows has promoted a scheme to identify the type of a file through the file name extension of the file. (The extension part of a file name is everything following the last dot in the file name.) Different types of files are then associated with the program that uses the file. When a file type and its association are made known to the operating system, when the user interacts with a known file type, the operarting system will start the program that uses the file directly. For instance, text files can be associated with the user's favorite editor. Double-clicking on a text file will then start the editor will the file ready to edit.

The *Configure ooDialog Services* dialog is used to start the process of configuring file associations for ooDialog.exe:


Configure ooDialog Services
Configure File Associations
For Current User
For All Users
Configure
Process Execution Status:
Done

User is in admin group: True
Process is run as admin: True
Process is elevated: True
Integrity Level: High

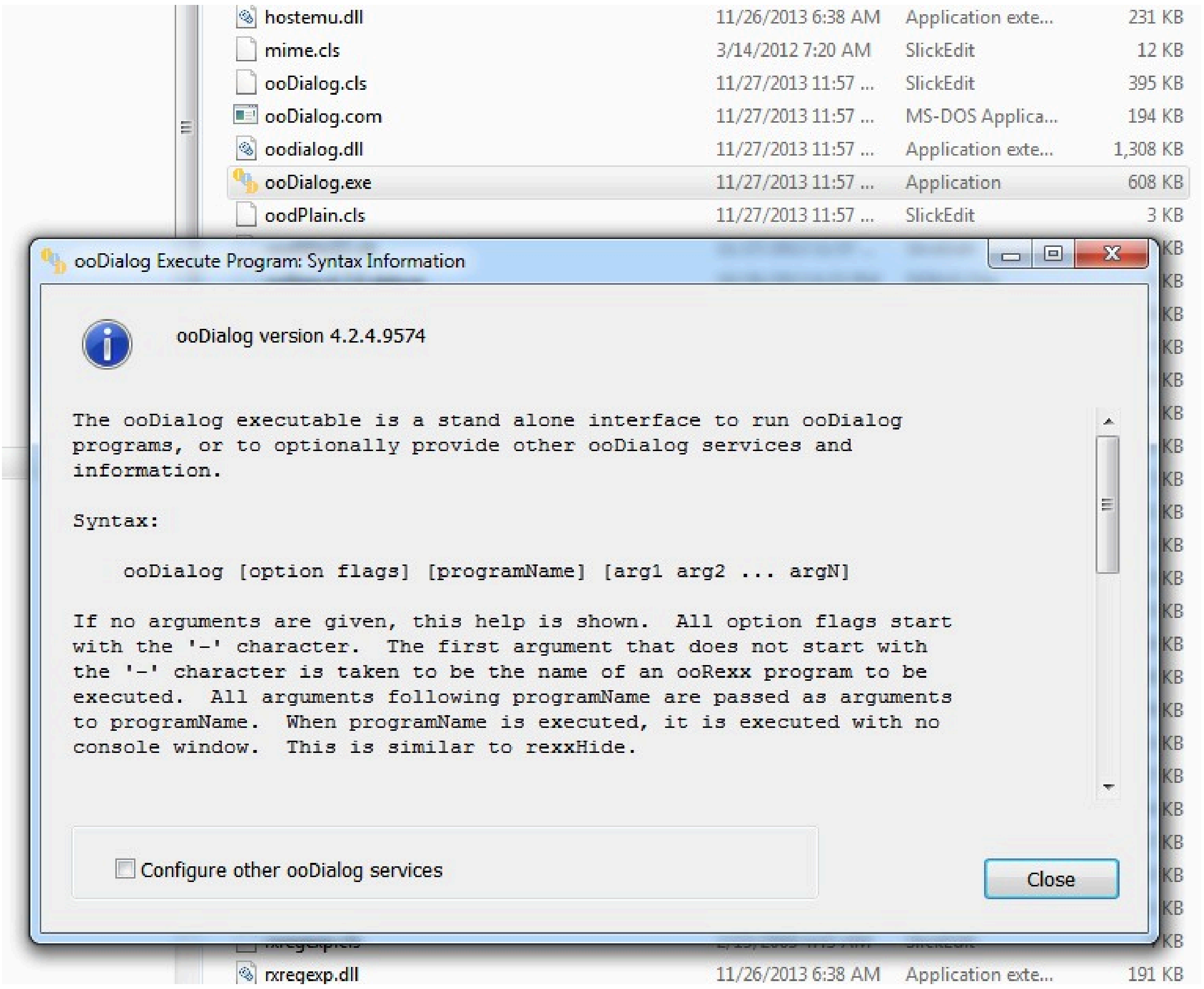
Figure 2.3. Configure Services Dialog

Future versions of ooDialog may have the ability to configure more services than just file associations, in which case this dialog will be expanded to allow that configuration. Currently, only file associations are configured through this dialog.

The configure services dialog can be brought up in two ways. From a command prompt, the *-s* option will immediately start the dialog:

```
C:\Rexx>oodialog -s

C:\Rexx>
```

Starting ooDialog.exe with no arguments, either from the command prompt or by double-clicking ooDialog.exe, brings up the information display window:


hostemu.dll 11/26/2013 6:38 AM Application exte... 231 KB
mime.cls 3/14/2012 7:20 AM SlickEdit 12 KB
ooDialog.cls 11/27/2013 11:57 ... SlickEdit 395 KB
ooDialog.com 11/27/2013 11:57 ... MS-DOS Applica... 194 KB
oodialog.dll 11/27/2013 11:57 ... Application exte... 1,308 KB
ooDialog.exe 11/27/2013 11:57 ... Application 608 KB
oodPlain.cls 11/27/2013 11:57 ... SlickEdit 3 KB
rxregexp.dll 11/26/2013 6:38 AM Application exte... 191 KB
ooDialog Execute Program: Syntax Information
ooDialog version 4.2.4.9574
Configure other ooDialog services
Close

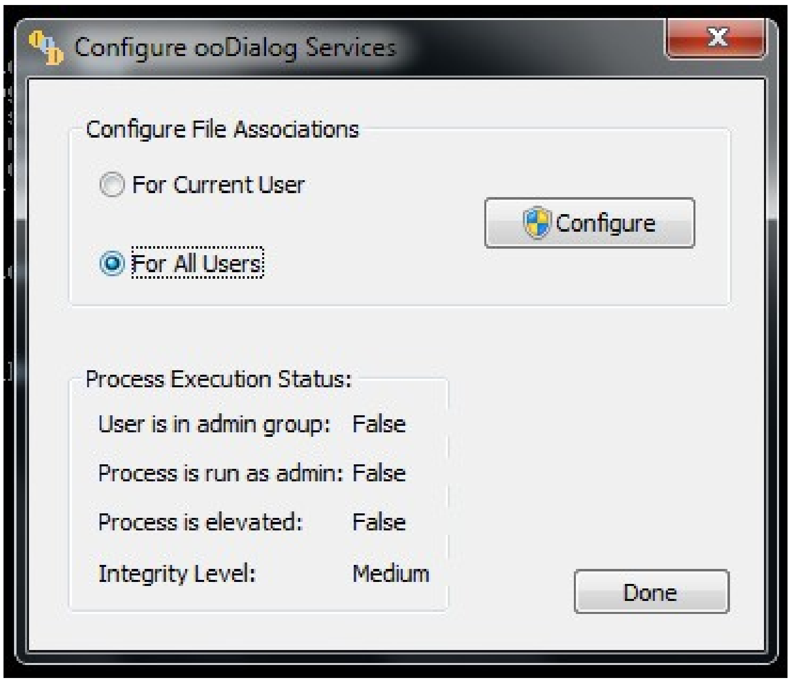
```
The ooDialog executable is a stand alone interface to run ooDialog
programs, or to optionally provide other ooDialog services and
information.

Syntax:

    ooDialog [option flags] [programName] [arg1 arg2 ... argN]

If no arguments are given, this help is shown.  All option flags start
with the '-' character.  The first argument that does not start with
the '-' character is taken to be the name of an ooRexx program to be
executed.  All arguments following programName are passed as arguments
to programName.  When programName is executed, it is executed with no
console window.  This is similar to rexxHide.
```

Figure 2.4. ooDialog Information Window

Check the *Configure other ooDialog services* check box. The *Close* button's label will change to *Continue* and clicking the button starts the configure services dialog.

The configure services dialog can be used to configure file type associations and the PATHEXT environment variable for either the current user, or for all users. Any user can configure file associations or environment variables for herself or himself. With the heightened security procedures introduced in Windows Vista and later, users can no longer set configurations of all users, unless they do so from an elevated process. By default the configure services dialog will be set to configure for the current user. If this is changed to all users, and the current process is not elevated, the UAC shield icon will be placed on the *Configure* button to indicate that the task requires administrator credentials. If *Configure* is clicked, the Credential UI will be displayed to obtain the credentials.

This is shown below. Note that the lower left quadrant of the dialog displays the security status of the running process. In this case the process is shown as *not* elevated. Changing the All Users file associations would required that the process be elevated.



Process Execution Status:

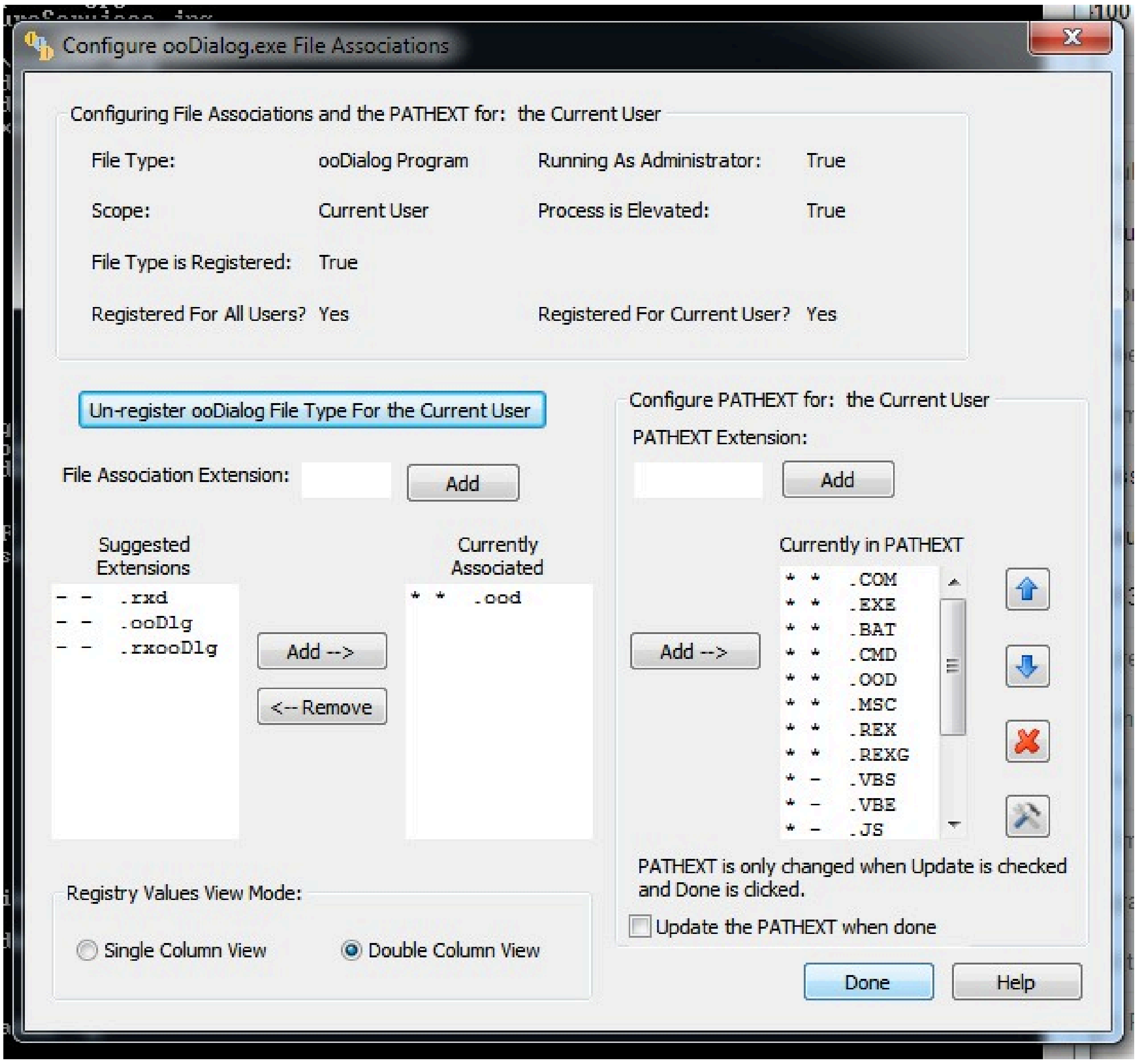
User is in admin group: False
Process is run as admin: False
Process is elevated: False
Integrity Level: Medium

Figure 2.5. Elevation Required

When the *For All Users* radio button was selectd, the UAC shield icon was added to the *Configure* button. If the user of the dialog changes the selected radio button to the *For Current User* button, the shield icon would be removed. At this point, if the dialog user clicks the *Configure* button, the Credential UI will be displayed. If the user enters the proper password for an Administrator, the Configure File Associations is displayed. Otherwise, it is not displayed.

Once the *scope* (the current user or all users) of the task is selected using the radio buttons, the user of the dialog proceeds by clicking the *Configure* button. As hinted at, the user of the dialog can only work with one scope at a time. However, when the selected scope is configured, the user will be returned to the configure user services dialog, and could then select to configure the other scope if desired.

This figure shows the dialog that is used to configure file associations for ooDialog.exe and the use of the dialog is explained in the following text:


Configure ooDialog.exe File Associations

Configuring File Associations and the PATHEXT for: the Current User

File Type:	ooDialog Program	Running As Administrator:	True
Scope:	Current User	Process is Elevated:	True
File Type is Registered:	True		
Registered For All Users?	Yes	Registered For Current User?	Yes

Un-register ooDialog File Type For the Current User
File Association Extension:
Add
Suggested Extensions
Currently Associated

```
- -   .rxd
- -   .ooDlg
- -   .rxooDlg
```

```
* *   .ood
```

Add -->
<-- Remove
Configure PATHEXT for: the Current User
PATHEXT Extension:
Add
Currently in PATHEXT
Add -->

```
* *   .COM
* *   .EXE
* *   .BAT
* *   .CMD
* *   .OOD
* *   .MSC
* *   .REX
* *   .REXG
* -   .VBS
* -   .VBE
* -   .JS
```

PATHEXT is only changed when Update is checked and Done is clicked.

☐ Update the PATHEXT when done

Registry Values View Mode:

○ Single Column View [✓ ● Double Column View]

Figure 2.6. Drag and Drop Scenario

To better explain how to use the dialog to configure the file associations, and or the PATHEXT, each part of the dialog will be discussed in turn. **Note:** it is important to keep in mind that file associations for all users and the current user are merged together by the operating system. Therefore the file associations in effect for any user are those registered for all users and those registered just for herself or himself. However, the operating system does not merge the PATHEXT. If there is a PATHEXT defined for the current user, the operating system *replaces* the all users PATHEXT with the PATHEXT defined by the current user.

Upper Third Portion:

The upper third of the dialog is an informational display only. It shows the scope of the current task, the process execution status, if ooDialog.exe is, or is not, registered as a file type, etc..

The Register Button:

The large register button is used to register, or un-register, the ooDialog program file type. The label of the button changes to reflect the current status of the registration. If the file type is currently registered, the button is used to un-register it. If the file type is not registered, the button is used to register the file type.

The registration or un-registration takes place immediately when the button is clicked. If the ooDialog program type is un-registered, all extensions associated with the file type, for the current scope, are un-associated at the same time. File extensions associated in the other scope can not be changed, the dialog can only work with one scope at a time.

File Association Extension / Add:

This edit box and push button are used to add a file extension to the currently associated types. Type the extension in the edit box and click the *Add* button. The association takes place immediately. The extension can be entered with or without the dot (`.`) character.

File Association List Boxes:

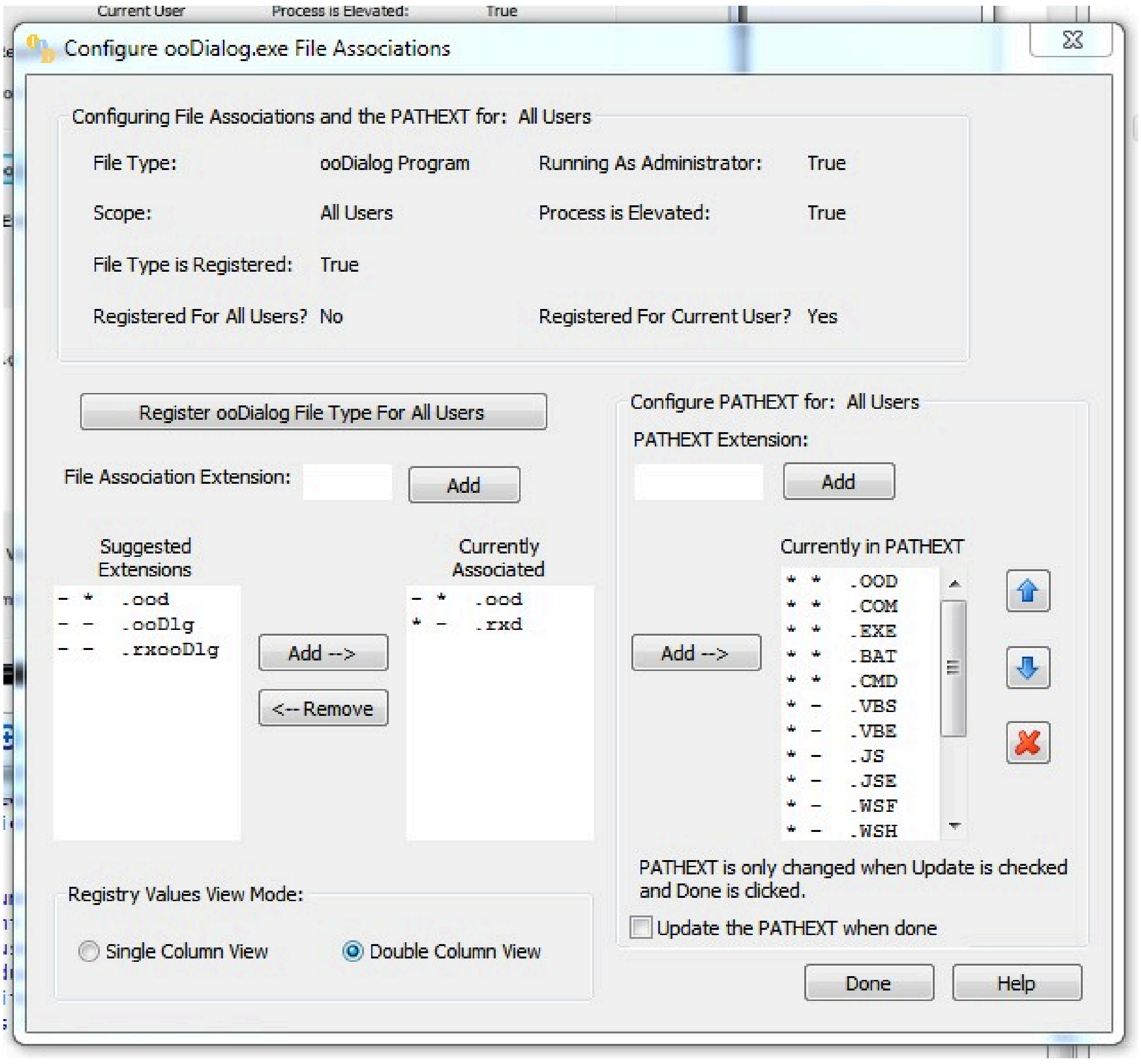
There are two list boxes used to display file association file extensions. The *Currently Associated* list box shows all file extensions associated with the ooDialog Program file type. The *Suggested Extensions* list box is initially filled with a few suggested extensions. This is provided as a convenience for the user of the dialog, there is no requirement that suggested extensions be used.

The list boxes can be in either dual column or single column mode. Recall that the operating system merges the file associations for all users and the current user. Because of this, if the ooDialog program file type is registered in either scope, (all users or the current user,) any associated file extension will work, no matter what scope it is associated in. This is reflected in the dual column mode. Every file extension in the currently associated list box is in effect for the user of the dialog. Each item in the list box has two columns. The first column shows the status for All Users, the second column shows the status for the Current User.

The **+** character is used to show that the extension is associated in that scope. The **-** character is used to show that the extension is not registered in that scope.

The *dual column* figure shows how to interpret the information displayed in the dialog. The Scope field shows that the task is currently configuring for All Users. The File Type is Registered field shows that the ooDialog Program file type is registered. The following fields show the file type is not registered for all users, but is registered for the current user. That is, the file type is not registered for the current scope of the task.

The items in the currently associated list box show the *ood* extension is not associated for all users / is associated for the current user, (**`- * .ood`**). The *rxd* extension is the inverse, associated for all users, not associated for the current user (**`* - .rxd`**).



Current User Process is Elevated: True

Configure ooDialog.exe File Associations

Configuring File Associations and the PATHEXT for: All Users

File Type:	ooDialog Program	Running As Administrator:	True
Scope:	All Users	Process is Elevated:	True
File Type is Registered:	True		
Registered For All Users?	No	Registered For Current User?	Yes

```
- *   .ood
- -   .ooDlg
- -   .rxooDlg
```

```
- *   .ood
* -   .rxd
```

```
* *   .OOD
* *   .COM
* *   .EXE
* *   .BAT
* *   .CMD
* -   .VBS
* -   .VBE
* -   .JS
* -   .JSE
* -   .WSF
* -   .WSH
```

PATHEXT is only changed when Update is checked and Done is clicked.

☐ Update the PATHEXT when done

Registry Values View Mode: Single Column View [✓ Double Column View]

Figure 2.7. Dual Column Example

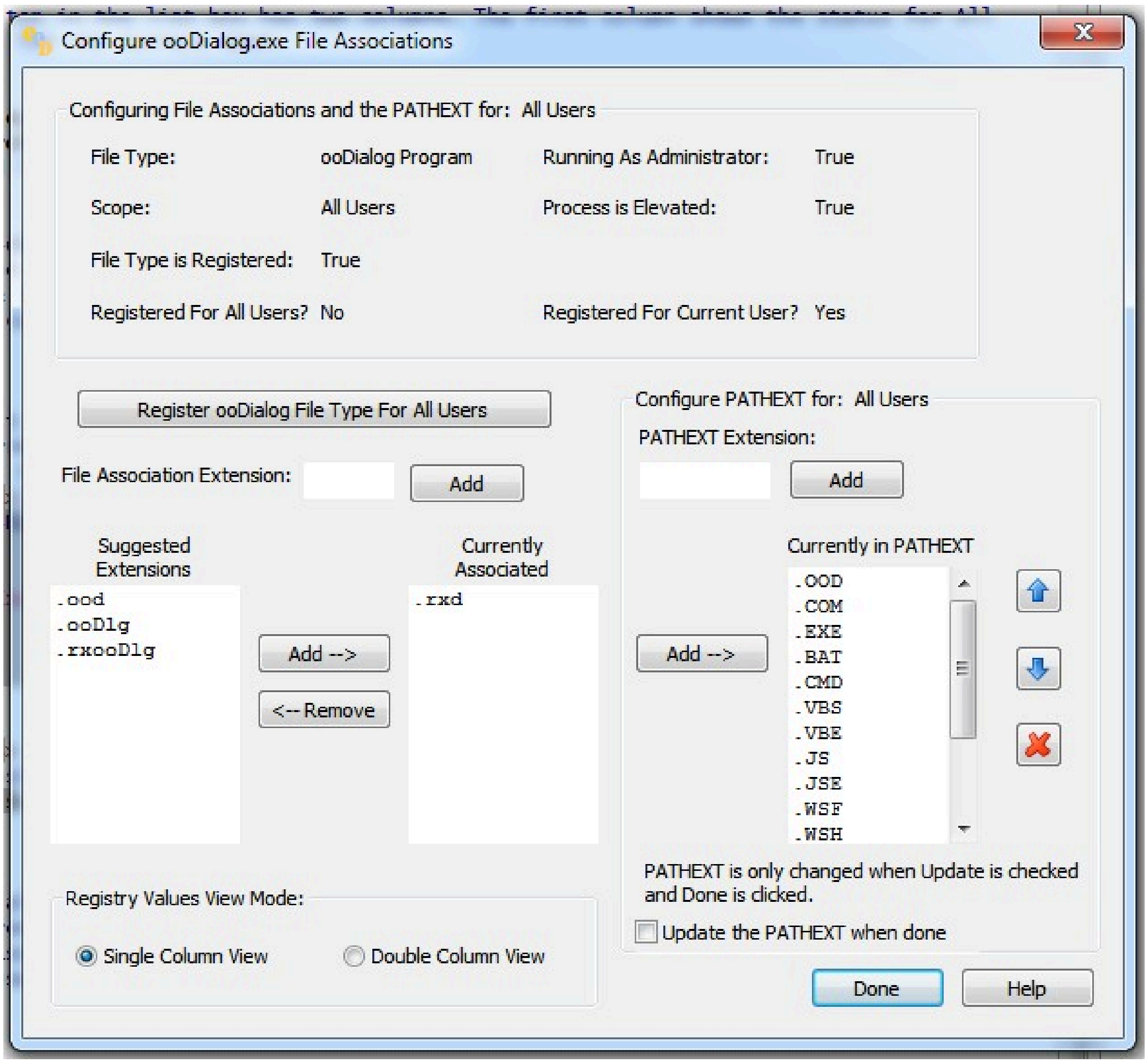
The *Add* and *Remove* buttons between the two list boxes are used to move the selected extension in one list box to the other. If an extension is selected in the *Suggested Extensions* list box and the add button clicked, the extension is immediately associated, in the current scope.

Likewise, if an extension is selected in the *Currently Associated* list box and the *Remove* button is clicked then the extension is immediately un-associated. The *Remove* button is the only way to un-associate an extension using the dialog.

Bear in mind that actions taken in the dialog are always only in relation to the current scope. For example in the figure, if the .ood extension in the currently associated list box is selected and the remove button is clicked, nothing will happen. The .ood extension is not associated in the current scope, so it can not be un-associated. The .ood extension is associated for the current user, but the dialog is not working on that scope, so again that can not be changed.

Single / Double Column View:

These two radio buttons are used to switch between dual column view, explained above, and single column view. In single column view the current registry values are shown only as they relate to the current scope. If the *dual column* figure were to be switched to single column view, it would appear like this:


Configure ooDialog.exe File Associations

Configuring File Associations and the PATHEXT for: All Users

File Type:	ooDialog Program	Running As Administrator:	True
Scope:	All Users	Process is Elevated:	True
File Type is Registered:	True		
Registered For All Users?	No	Registered For Current User?	Yes

Register ooDialog File Type For All Users

File Association Extension: Add

Suggested Extensions: .ood, .ooDlg, .rxooDlg

Currently Associated: .rxd

Configure PATHEXT for: All Users

PATHEXT Extension: Add

Currently in PATHEXT: .OOD, .COM, .EXE, .BAT, .CMD, .VBS, .VBE, .JS, .JSE, .WSF, .WSH

PATHEXT is only changed when Update is checked and Done is clicked.

Update the PATHEXT when done

Registry Values View Mode: [✓ Single Column View] Double Column View

Figure 2.8. Single Column Example

Note that the .ood extension is not associated for all users, so it does not appear in the currently associated list box. In some ways the single column view may be less confusing. But, for file associations, it does not present a true picture. The .ood extension is listed in the suggested extensions list box, but there is no real reason to associate it. Since .ood is associated for the current user, when the operating system merges the file associations for all users and the current user, it becomes associated in effect. Switching between the two views may be the best way to understand the current associations.

PATHEXT related dialog controls:

The PATHEXT related dialog controls allow the user of the dialog to add or remove extensions from the PATHEXT environment variable, and / or to re-order the extensions in PATHEXT. The operating system uses this environment variable to determine if a file is an executable file from its name. If a file's name has an extension in the PATHEXT, then the operating system treats the file as an executable program rather than a data file. Files whose extension is in the PATHEXT are also executed by typing their file name, without the extension, on the command prompt.

This functionality of editing the PATHEXT variable is added as a convenience for the dialog user. The behavior of this functionality is similar to the functionality of configuring the file type associations for ooDialog.exe, but also has some differences. Keep the following points in mind when working with the PATHEXT in the dialog:

- The purpose of editing PATHEXT in the dialog is to allow the dialog user to easily add an associated extension to the PATHEXT. For instance if the extension **`.ood`** is associated with the ooDialog file type, then adding that extension to the PATHEXT will allow any ooDialog programs with the extension to be executed by ooDialog.exe automatically if the file name, with or without the extension, is typed on the command line.

- Unlike file association changes which take place immediately, no changes to the PATHEXT are saved until the *Done* button is clicked. In addition, the *Update the PATHEXT when done* check box must be checked at the time the button is clicked. This is done to ensure the dialog user really intends for the changes to be made.

- Unlike file associations, the operating system does not merge the All Users PATHEXT and the Current User's PATHEXT values. If the current user sets a PATHEXT value, it *replaces* the All Users PATHEXT for that user.

- A poorly formed PATHEXT value can cause the system to behave erratically. For instance if PATHEXT did not contain the **`.EXE`** extension, programs might not be able to be executed. For this reason, the dialog will not remove the .exe, .cmd, .com, and .bat extensions from PATHEXT. The only exception to this is if the scope is the Current User, the dialog will allow the user to completely delete the PATHEXT variable for the Current User. The All Users PATHEXT can not be deleted through the dialog.

- Like the file associations list boxes, the PATHEXT list box can be displayed in dual or single column view. The *description* for the File Association List Boxes explains how the two views work and how to interpret the list box items. However, since the operating system does not merge the All Users and Current Users PATHEXT values, the single column view may be easier to use when editing the PATHEXT.

PATHEXT Extension / Add:

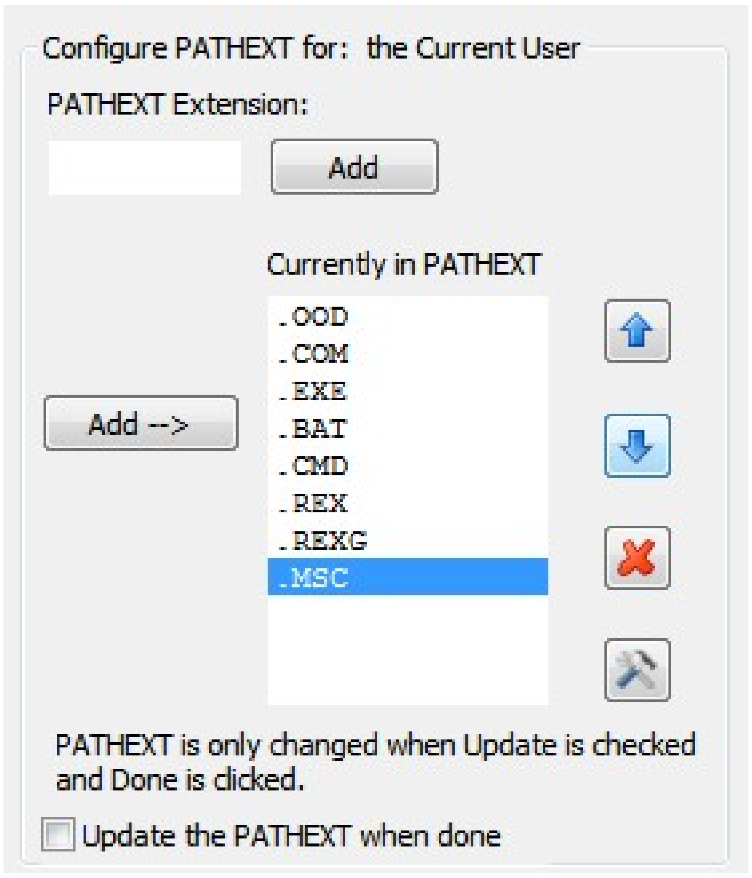
The PATHEXT edit box and Add button are similar in use to the File Association Extension edit box and Add button. The dialog user types the extension in the edit box and clicks the *Add* button to add a file extension to the PATHEXT list box for the current scope. The dot character may or may not be used. If the extension entered in the edit box does not start with a dot character, one is added. The extension is also upper-cased. Other than that, no attempt is made to validate the entry typed by the dialog user.

PATHEXT List Box:

The PATHEXT list box is initially filled with information that depicts the current PATHEXT for All Users and the PATHEXT for the Current User. The Current User PATHEXT may not exist, in fact that is relatively common. The dialog user can make changes to the PATHEXT in the current scope by interacting with the PATHEXT list box and its related buttons. The list box will always reflect how the PATHEXT will be saved, if the *Update the PATHEXT when done* check box is checked and the *Done* button is pressed. The *description* for the File Association List Boxes describes how to interpret the list box items. For the PATHEXT, it is probably easier to interpret what the PATHEXT value will be saved as in single column view.

PATHEXT List Box Buttons:

This figure shows the buttons surrounding the PATHEXT list box that are used to edit the PATHEXT


Configure PATHEXT for: the Current User

PATHEXT Extension:

Add

Currently in PATHEXT

.OOD
.COM
.EXE
.BAT
.CMD
.REX
.REXG
.MSC

Add -->

PATHEXT is only changed when Update is checked and Done is clicked.

☐ Update the PATHEXT when done

Figure 2.9. PATHEXT Editing Buttons

The figure shows that the current scope of the dialog is the Current User. The PATHEXT list box is in single column view, so the list box items reflect exactly which extensions, and their order, will be written to the PATHEXT for the Current User, if the user were to save the PAThEXT.

The upper *Add* button has already been described. If the user has typed an extension in the edit box, the *Add* button will add that extension to the list box. The *Add* --> button on the left serves a similar function. If an item in the *Currently Registered* list box is selected, clicking the *Add* --> will add it to the list box.

The Up and Down arrow buttons are used to re-order the extensions. If an item in the list box is selected, clicking the Up arrow button will move the item up one position. Clicking the Down arrow button moves the item down one position. Clicking the X button deletes the currently selected item from the PATHEXT.

The Hammer and Cresent Wrench button is only present when the scope is the Current User. Clicking this button deletes the Current Users PATHEXT completely.

Update the PATHEXT when done Check Box:

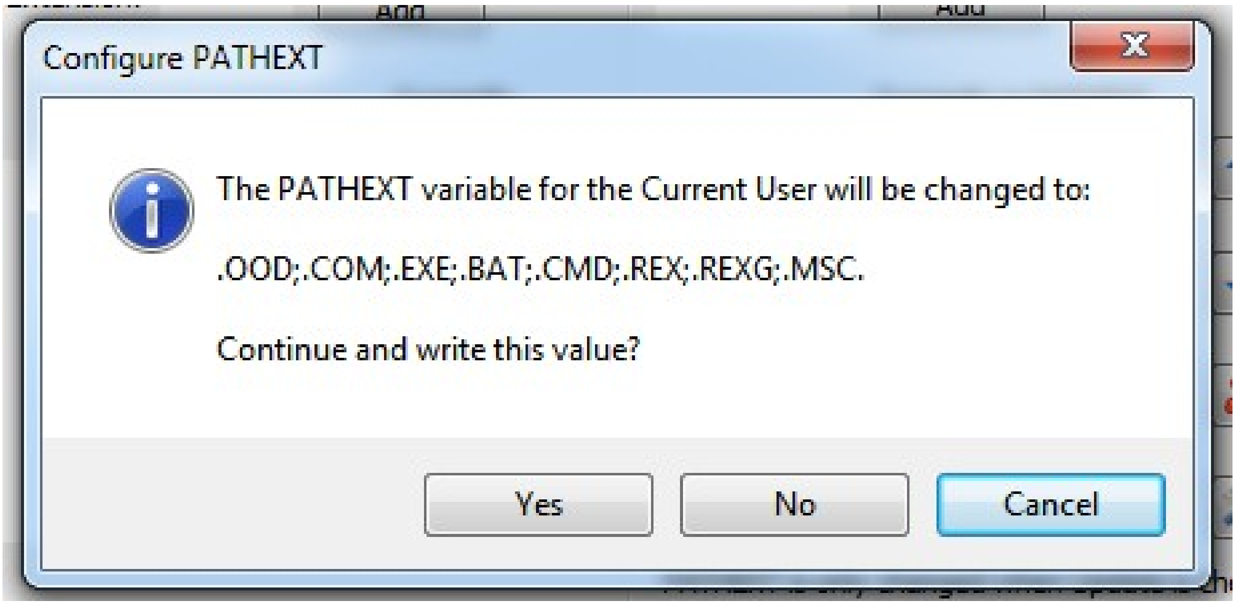
As mentioned above, the PATHEXT is not changed, not updated, until the dialog is closed with the *Done* button. If and only if the *Update the PATHEXT when done* check box is checked, the new PATHEXT value, as defined by the dialog user while interacting with the dialog, is written to the registry.

Done Button:

Clicking the *Done* button closes the dialog. Since all file association changes are written to the registry as the dialog user makes them, clicking the *Done* button and using the **Esc** key to close the dialog are essentially the same. The only time using the *Done* button is of importance is if the dialog user wants changes to the PATHEXT to be saved. In this case, the *Update the PATHEXT* *when done* check box must be checked and the *Done* button must be used to close the dialog. If the check box is checked, but the user closes the dialog with the **Esc** key, any PATHEXT changes are discarded.

When the *Done* button is clicked, before the new PATHEXT is actually written, a message box will display exactly how the PATHEXT will saved. This message box gives the user of the dialog one last chance to confirm that she wants the PATHEXT to be changed.

Clicking *Yes* writes the new PATHEXT value to the registry. Clicking *No* discards any changes and closes the dialog. Clicking *Cancel* returns to the configure file associations dialog allowing the dialog user to retry making changes to PATHEXT.


Configure PATHEXT

The PATHEXT variable for the Current User will be changed to:

.OOD;.COM;.EXE;.BAT;.CMD;.REX;.REXG;.MSC.

Continue and write this value?

Figure 2.10. Change PATHEXT Confirmation

The Dialog Object

All instantiated Rexx dialog objects have a broad base of methods in common. These methods are the same for all dialogs regardless of the class of the dialog or how the dialog *template* is created.

The dialog object here is an abstract concept, used to make it simpler to understand the basic concepts. To instantiate a dialog object you create a subclass of one of the concrete dialog classes provided by the ooDialog framework, such as the UserDialog *UserDialog*, ResDialog *ResDialog*, or RcDialog *RcDialog*. This chapter provides the reference for the constants, attributes, and methods that are common to all dialog objects. See the specific dialog subclass for methods that are unique to the subclass.

3.1. Method Table

The following table lists the constant methods, class methods, attribute methods, and instance methods that all dialogs have in common:

Table 3.1. Dialog Object Method Reference

Dialog Method	Description
Constant Methods	The dialog object provides a number of *constant* values through the `::constant`directive.
Class Methods	**Class Methods**
new	Instantiates a new dialog object.
getFontName	Returns the default font for all dialogs.
getFontSize	Returns the default font size for all dialogs.
setDefaultFont	Sets the default font name and size for all dialogs.
Attribute Methods	**Attribute Methods**
autoDetect	If automatic data field detection is on or off.
constDir	A directory object that maps symbolic resource IDS to their numeric IDs
dlgHandle	The window handle of the dialog.
dlgID	The *dlgID* attribute reflects a whole number value that can be used to identify this dialog.
factorX	The horizontal size of one dialog unit in pixels. (Inaccurate.)
factorY	The vertical size of one dialog unit in pixels. (Inaccurate.)
fontName	Name of the dialog's font.
fontSize	Point size of the dialog's font.
hwnd	The window handle of the dialog.
initCode	Reflects if the object initialization was successful.
ownerDialog	The owner dialog of the dialog, if there is one.
pixelCX	The width of the dialog in pixels.
pixelCY	The height of the dialog in pixels.
sizeX	The width of the dialog in dialog units. (Inaccurate.)
sizeY	The height of the dialog in dialog units. (Inaccurate.)
Instance Methods	**Instance Methods**

Dialog Method	Description
addAutoStartMethod	Adds a method and its arguments to be started automatically and run concurrently when the dialog is executed.
addComboEntry	Adds a string to the list of a combo box.
addListEntry	Adds a string to the specified list box.
addNewAttribute	Adds a new attribute (a method) to this dialog.
addNewMethod	Adds a new method to this dialog.
addUserMsg	Connects an operating system window message with a method in the Rexx dialog object.
autoDetection	Turns automatic data field detection on
backgroundBitmap	Sets a bitmap as the dialog's background picture.
backgroundColor	Sets the background color of a dialog.
backgroundSysColor	Sets the background color of a dialog using one of the system colors.
cancel	A default event handler, provided by the ooDialog framework , for the cancel event.
createBrush	Creates a logical brush that has the specified style, color, and pattern.
center	Moves the dialog to the screen center.
changeBitmapButton	Changes the bitmaps for an already installed bitmap button and optionally immediately redraws the button.
changeComboEntry	changeComboEntry
changeListEntry	changeListEntry
childWindowFromPoint	Determines which, if any, of the child windows belonging to this dialog contains the specified point.
clear	Clears the client area of the dialog by painting it with the background brush.
clearControlRect	Clears the client area of the specified dialog control by redrawing the area with the background brush set to the default background color of a dialog.
clearRect	Clears the specified rectangular within the client area of a window.
clearWindowRect	Erases the client area of the given window.
client2screen	Converts a point, or points, in client-area coordinates of the dialog to its screen coordinates.
clientRect	Returns a **`Rect`** object containing the dimensions of the dialog's client area in pixels.
clientToScreen	Converts client-area coordinates of the dialog to its screen coordinates.
comboAddDirectory	Adds all or selected file names in the given directory to the combo box.
comboDrop	Deletes all items from the list of the given combo box.
connectActivate	Connects the window activation event to a method in the Rexx dialog.
connectAllSBEvents	Connects all event notifications from a scroll bar control to a single method in the Rexx dialog.
connectButtonEvent	Connect an event notification from a button control to a method in the Rexx Dialog

Dialog Method	Description
connectCheckBox	Creates a data attribute in the Rexx dialog object and connects it to a check box control in the underlying dialog.
connectComboBox	Creates a data attribute in the Rexx dialog object and connects it to a combo box control in the underlying dialog.
connectComboBoxEvent	Connects an event notification from a combo box to a method in the Rexx dialog.
connectCommandEvents	Connects a command event notification from a dialog control to a method in the Rexx dialog.
connectDateTimePicker	Creates a data attribute in the Rexx dialog object and connects it to a date time picker control in the underlying dialog.
connectDateTimePickerEvent	Connects an event notification form a date time picker to a method in the Rexx dialog.
connectDraw	Connects the draw item event notification to a method in the Rexx dialog.
connectEachSBEvent	Connects each specified event notification from a scroll bar to a separate method in the Rexx dialog.
connectEdit	Creates a data attribute in the Rexx dialog object and connects it to a edit control in the underlying dialog.
connectEditEvent	Connects an event notification from an edit control to a method in the Rexx dialog.
connectFKeyPress	Connects a F Key key press (a F key is typed) with a method in the Rexx dialog.
connectHelp	Connects the Windows Help event to a method in the Rexx dialog.
connectKeyPress	Connects a key press (a key is typed) with a method in the Rexx dialog.
connectListBox	Creates a data attribute in the Rexx dialog object and connects it to a list box control in the underlying dialog.
connectListBoxEvent	Connects an event notification from a list box control to a method in the Rexx dialog.
connectListView	Creates a data attribute in the Rexx dialog object and connects it to a list view control in the underlying dialog.
connectListViewEvent	Connects an event notification from a list-view control to a method in the Rexx dialog.
connectMonthCalendar	Creates a data attribute in the Rexx dialog object and connects it to a month calendar control in the underlying dialog.
connectMonthCalendarEvent	Connects an event notification from a month calendar to a method in the Rexx dialog.
connectMove	Connects the move event notification to a method in the Rexx dialog.
connectNotifyEvent	Connects a generic event notification from a dialog control to a method in the Rexx dialog.
connectPosChanged	Connects the position has changed event notification to a method in the Rexx dialog.
connectRadioButton	Creates a data attribute in the Rexx dialog object and connects it to a radio button control in the underlying dialog.
connectResize	Connects the size event notification to a method in the Rexx dialog

Dialog Method	Description
connectResizing	Connects the sizing event notification to a method in the Rexx dialog
connectScrollBarEvent	Connects an event notification from a scroll bar control to a method in the Rexx dialog.
connectSizeMoveEnded	Connects the size / move ended event notification to a method in the Rexx dialog object.
connectStaticEvent	Connects an event notification from a static control to a method in the Rexx dialog.
connectTab	Creates a data attribute in the Rexx dialog object and connects it to a tab control in the underlying dialog.
connectTabEvent	Connects an event notification from a tab control to a method in the Rexx dialog.
connectToolBarEvent	Connects an event notification from a toolbar control to a method in the Rexx dialog.
connectToolTipEvent	Connects an event notification from a tool tip control to a method in the Rexx dialog.
connectTrackBar	Creates a data attribute in the Rexx dialog object and connects it to a track bar control in the underlying dialog.
connectTrackBarEvent	Connects an event notification from a track bar control to a method in the Rexx dialog.
connectTreeView	Creates a data attribute in the Rexx dialog object and connects it to a tree view control in the underlying dialog.
connectTreeViewEvent	Connects an event notification from a tree view control to a method in the Rexx dialog.
connectUpDown	Creates a data attribute in the Rexx dialog object and connects it to a up-down control in the underlying dialog.
connectUpDownEvent	Connects an event notification from an up-down control to a method in the Rexx dialog.
createBrush	Creates a logical brush that has the specified style, color, and pattern.
createFont	Returns a handle to a logical font, the implementation is **incorrect**.
createFontEx	Retrieves a handle to a logical font from the system font manager
createPen	Creates a logical pen that has the specified style, width, and color.
createToolTip	Creates the underlying Windows ToolTip control and returns an instantiated Rexx *ToolTip* object.
defListDragHandler	Default implementation of a drag and drop handler for a list-view control.
defTreeDragHandler	Default implementation of a drag and drop handler for a tree view control.
deleteComboEntry	Deletes a string from the combo box.
deleteFont	Deletes a font returned from *createFontEx* or *createFont*.
deleteListEntry	Deletes an item from a list box.
deleteObject	Deletes a graphic object,
determineSBPosition	Calculates and sets the new scroll bar position based on the position data retrieved from the scroll bar and the step information.
dimBitmap	Draws a bitmap step by step.

Dialog Method	Description
disable	Disables the dialog.
disableControl	Disables the dialog control with the specified ID.
disconnectKeyPress	Disconnects a method that was previously connected to key press event.
displaceBitmap	Sets the position of the bitmap within the client area of a bitmap button.
display	Shows or hides the dialog.
dlgUnit2pixel	Takes a dimension expressed in dialog units of this dialog and transforms it to a dimension expressed in pixels.
draw	Redraws the entire client area of the dialog immediately.
drawAngleArc	Draws a partial circle (arc) and a line connecting the start drawing point with the start of the arc.
drawArc	Draws a circle or ellipse withi the given device context using the active pen.
drawBitmap	Draws all, or part of, the bitmap for a bitmap button at the specified position.
drawButton	Draws the specified button.
drawLine	Draws a line within the device context using the active pen.
drawPie	Draws and fills a pie of a circle or ellipse within the given device context using the active brush and pen.
drawPixel	Draws a pixel within the device context.
enable	Enables the dialog.
enableControl	Enables the control with the specified resource id.
endAsyncExecution	Used to end the asynchronous execution of a dialog started using the executedAsync method.
ensureVisible	Causes the dialog to reposition itself so that it is entirely on the visible screen.
execute	Creates the underlying dialog, shows it, starts the automatic methods, if any, and destroys the underlying dialog when the user closes it.
executeAsync	Creates the underlying Windows dialog, starts it executing, shows it in the same way as the execute method does, and returns immediately.
fillDrawing	Fills in an outline figure within the device context using the active brush.
fillRect	Fills a rectangle using the specified brush within the specified device context.
findComboEntry	Returns the index corresponding to a given text string in the combo box.
findListEntry	Returns the index of the specified string within the list box with the specified ID.
focusControl	Sets the input focus to the specified dialog control.
fontColor	Sets the font color for a device context.
fontToDC	Loads a font into a device context and returns the handle of the previous font.
foregroundWindow	Returns the handle of the window in the foreground.
freeDC	Releases a device context.

Dialog Method	Description
freeControlDC	Releases the device context of a control.
freeWindowDC	Releases the device context of a window.
get	Returns the window handle of the Windows dialog associated with the top Rexx dialog instance.
getArcDirection	Returns the current drawing direction.
getBitmapPosition	Gets the position, as a **`Point`** object, of the bitmap in a bitmap button.
getBitmapSizeX	Gets the width in pixels of the bitmap assigned to a bitmap button.
getBitmapSizeY	Gets the height in pixels of the bitmap assigned to a bitmap button.
getBmpDisplacement	Gets the position, as a **`String`** object, of the bitmap in a bitmap button.
getCheckBoxData	Sets the value of the connected data attribute in the Rexx dialog object to match the state of the specified check box in the underlying dialog.
getClientRect	Returns the dimensions of the dialog's client area.
getComboBoxData	Sets the value of the connected data attribute in the Rexx dialog object to match the state of the specified combo box in the underlying dialog.
getComboEntry	Returns the string at the specified index of the combo box.
getComboItems	Returns the number of items in the combo box.
getControlData	Sets the value of the connected data attribute in the Rexx dialog object to match the state of the specified dialog control in the underlying dialog.
getControlDC	Returns the device context of a dialog control.
getControlHandle	Retrieves the window handle of the specified dialog control.
getControlID	Returns the numeric resource ID of the control with the specified window handle.
getControlRect	Returns the size and position rectangle of the specified dialog control.
getControlText	Gets the text of the specified dialog control
getCurrentComboIndex	Returns the index of the currently selected item in the comb box.
getCurrentListIndex	Returns the index of the currently selected list box item, or 0 if no item is selected.
getData	Sets all the Rexx dialog data attribute values to the state of the underlying, connected, Windows dialog controls.
getDataAttribute	Sets a data attribute of the Rexx dialog using the data from a connected Windows dialog control.
getDataStem	Sets the values of the indexes of the specified stem to the state of the underlying, connected, Windows dialog controls.
getDC	Returns the handle to the display device context of a dialog
getEditData	Sets the value of the connected data attribute in the Rexx dialog object to match the state of the specified edit control in the underlying dialog.
getExStyleRaw	Retrieves the numeric value of the dialog's extended style flags.
getFocus	Returns the window handle of the dialog control that currently has the input focus.
getFont	Returns the font used by the dialog.
getID	Retrieves the identification number of the dialog.

Dialog Method	Description
getListBoxData	Sets the value of the connected data attribute in the Rexx dialog object to match the state of the specified list box in the underlying dialog.
getListEntry	Returns the string at the specified index of the list box.
getListItemHeight	Returns the height of the items in a list box in dialog units. **(Inaccurate)**
getListItemHeightPx	Returns the height of the items in a list box in pixels.
getListItems	Returns the number of items in the list box.
getListWidth	Returns the scrollable width of a list box in dialog units. **(Inaccurate)**
getListWidthPx	Returns the scrollable width of a list box in pixels.
getMenuBar	Returns the menu object attached to the dialog, or .nil if there is no menu attached.
getPixel	Returns the color number of a pixel within the device context.
getPos	Returns the position of the dialog in dialog units **(not accurate.)**
getRadioButtonData	Sets the value of the connected data attribute in the Rexx dialog object to match the state of the specified radio button in the underlying dialog.
getRealPos	Returns the position of the dialog in pixels as a **`Point`** object.
getRealSize	Returns the size of the dialog in pixels as a **`Size`** object.
getRect	Returns the dimensions of the dialog.
getSBPos	Returns the current position of a scroll bar control.
getSBRange	Returns the range of a scroll bar control.
getSelf	Returns the window handle of the dialog.
getSize	Returns the size of the dialog in dialog units **(not accurate.)**
getStyleRaw	Retrieves the numeric value of the dialog's style flags.
getSysBrush	Retrieves a handle to a logical brush that corresponds to the specified system color index.
getText	Gets the text of the dialog.
getTextAlign	Gets the text alignment setting for the specified device context.
getTextExtent	Gets the bounding rectangle, as a *Size* object for the specified text, if it were to be drawn in the specified device context.
getTextSizeDlg	Calculates the size needed to display a string in dialog units.
getTextSizeDu	Calculates the size needed to display a string in dialog units **(preferred method.)**
getTextSizePx	Calculates the size needed for a string in pixels **(preferred method.)**
getTextSizeScreen	Calculates the size needed for a string in pixels.
getTextSizeTitleBar	Gets the size, width and height, of a string used in the caption bar for the title.
getWindowDC	Returns the device context of a window.
getWindowText	Gets the text of the specified window.
hasKeyPressConnection	Queries if a connection to a key press event already exists.
hasMenuBar	Tests if the dialog has a menu bar attached.
help	A default implementation of the help event handler method provided by the ooDialog framework.

Dialog Method	Description
hide	Hides the dialog by marking it invisible and immediately redrawing the area it occupies.
hideFast	Marks the dialog as invisible, no redrawing is done.
hideControl	Hides the specified control by marking it invisible and immediately redrawing the area it occupies.
hideControlFast	Marks the specified control as invisible without any redrawing.
hideWindow	Hides a whole dialog window or a dialog control window and forces the window to repaint.
hideWindowFast	Hides a whole dialog window or a dialog control window, but does not force the window to redraw.
hScrollPos	Returns the position of the horizontal scroll bar in the dialog.
initAutoDetection	Switches automatic data field detection on or off.
initDialog	A method automatically invoked by the ooDialog framework when the underlying dialog is first created.
installAnimatedButton	Installs an animated button and runs it concurrently with the main activity.
installBitmapButton	Connects bitmap(s) and a method with a push button.
insertComboEntry	Inserts a string into the list of a combo box.
insertListEntry	Inserts a string into the specified list box.
isDialogActive	Returns true if the underlying Windows dialog still exists.
isEnabled	Tests if the dialog is enabled.
isMaximized	Checks if a dialog is currently maximized.
isMinimized	Checks if a dialog is currently minimized.
isVisible	Tests if the dialog is visible.
leaving	A method automatically invoked by the ooDialog framework when the underlying dialog is being closed.
listAddDirectory	Adds all or selected file names of a given directory to the list box.
listDrop	Removes all items from the list box.
loadBitmap	Loads a bitmap from a file into memory and returns the handle to the bitmap.
mapWindowPoints	Converts, or maps, a set of points from the coordinate space relative to this dialog to a coordinate space relative to the specified window.
maximize	Maximizes the dialog on the screen.
minimize	Minimizes the dialog to the taskbar.
move	Moves the dialog to the position specified in dialog units **(not accurate.)**
moveControl	Moves a dialog control to another position within the dialog window.
moveTo	Moves the dialog to the position specified in pixels.
moveWindow	Changes the position, visibility, and Z order of the dialog.
newCheckBox	Returns an object of the **`CheckBox`** class for the check box control with the specified resource ID.

Dialog Method	Description
newComboBox	Returns an object of the **`ComboBox`** class for the combo box control with the specified resource ID.
newDateTimePicker	Returns an object of the **`DateTimePicker`** class for the date time picker control with the specified resource ID.
newEdit	Returns an object of the **`Edit`** class for the edit control with the specified resource ID.
newGroupBox	Returns an object of the **`GroupBox`** class for the group box control with the specified resource ID.
newListBox	Returns an object of the **`ListBox`** class for the list box control with the specified resource ID.
newListView	Returns an object of the **`ListView`** class for the list-view control with the specified resource ID.
newMonthCalendar	Returns an object of the **`MonthCalendar`** class for the month calendar control with the specified resource ID.
newProgressBar	Returns an object of the **`ProgressBar`** class for the progress bar control with the specified resource ID.
newPushButton	Returns an object of the **`Button`** class for the push button control with the specified resource ID.
newRadioButton	Returns an object of the **`RadioButton`** class for the radio button control with the specified resource ID.
newReBar	Returns an object of the **`ReBar`** class for the rebar control with the specified resource ID.
newScrollBar	Returns an object of the **`ScrollBar`** class for the scroll bar control with the specified resource ID.
newStatic	Returns an object of the **`Static`** class for the static control with the specified resource ID.
newStatusBar	Returns an object of the **`StatusBar`** class for the status bar control with the specified resource ID.
newTab	Returns an object of the **`Tab`** class for the tab control with the specified resource ID.
newToolBar	Returns an object of the **`ToolBar`** class for the toolbar control with the specified resource ID.
newToolTip	Returns an object of the **`ToolTip`** class for the tool tip control with the specified resource ID.
newTrackBar	Returns an object of the **`TrackBar`** class for the track bar control with the specified resource ID.
newTreeView	Returns an object of the **`TreeView`** class for the tree-view control with the specified resource ID.
newUpDown	Returns an object of the **`UpDown`** class for the up-down control with the specified resource ID.
noAutoDetection	Turns automatic data field detection off
objectToDC	Loads a graphic object into a device context.
ok	A default event handler, provided by the ooDialog framework, for the cancel event.

Dialog Method	Description
opaqueText	Sets the text drawing mode in a device context to opaque, (background is redrawn with the current brush.)
parseIncludefile	Reads a file and adds any symbol definitions found to the proper constant directory.
pixel2dlgUnit	Takes a dimension expressed in pixels and transforms it to a dimension expressed in dialog units of this dialog.
popup	Starts a modeless dialog executing and returns immediately.
popupAsChild	Assigns a parent dialog, starts a modeless dialog executing, and returns immediately. The dialog is closed automatically when the parent dialog ends.
resizeControl	Changes the size of a dialog control.
rectangle	Draws a rectangle within the given device context.
redraw	Redraws the entire dialog window and all its child windows immediately.
redrawClient	Redraws the entire client area of the dialog immediately.
redrawControl	Redraws the specified dialog control.
redrawRect	Redraws a rectangle within the client area of this dialog, or optionally the client area of a specified window.
redrawWindow	Redraws the specified window.
redrawWindowRect	Forces this dialog to completely redraw its client area, or optionally the client area of a specified window.
removeBitmap	Frees an in-memory bitmap that was loaded through the *loadBitmap* method.
resize	Resizes the dialog to the size specified in dialog units **(not accurate.)**
resizeTo	Resizes the dialog to the size specified in pixels.
resolveNumericID	Resolves a numeric ID to a symbolic ID, if the symbol exists.
resolveSymbolicID	Resolves a, possibly, symbolic ID to its numeric value.
restore	Restores a minimized or maximized dialog to its original position.
screen2client	Converts a point or points in screen coordinates to the client-area coordinates of the dialog.
screenToClient	Converts screen coordinates to the client-area coordinates of the dialog.
scroll	Scrolls the contents of the dialog's client area by the amount specified.
scrollBitmapFromTo	Scrolls the bitmap within the bitmap button from one position to another.
scrollButton	Moves a rectangle within the button and redraws the uncovered area with the button background color.
scrollInControl	Scrolls text in a dialog control using the specified font.
scrollText	Scrolls text in the specified window using the specified font.
sendMessage	Sends a Windows message to the underlying dialog and returns its response as a whole number.
sendMessageHandle	Sends a Windows message to the underlying dialog and returns its response as a handle.
sendMessageToControl	Sends a Windows message to a dialog control and returns its response as a whole number.

Dialog Method	Description
sendMessageToControlH	Sends a Windows message to a dialog control and returns its response as a handle.
sendMessageToWindow	Sends a Windows message to a window and returns its response as a whole number.
sendMessageToWindowH	Sends a Windows message to a window and returns its response as a handle.
setArcDirection	Sets the current drawing direction.
setBitmapPosition	Sets the position of the upper left corner of a bitmap within the bitmap button.
setCheckBoxData	Sets the *data* of a check box control.
setComboBoxData	Sets the *data* of a combo box control.
setControlColor	Sets the background color, and optionally the text color, for the specified dialog control.
setControlData	Sets the *data* of a dialog control.
setControlFont	Changes the font for specified dialog control.
setControlSysColor	Sets the background color, and optionally the text color, for the specified dialog control using the system colors.
setControlText	Sets the text for the specified dialog control
setCurrentComboIndex	Selects the item with the specified index within the combo box list. If called without an index, all items in the combo box list are deselected.
setCurrentListIndex	Selects the item with the specified index in the list box. If called without an index, all items in the list box are deselected.
setData	Sets the state of all underlying, connected, dialog controls to the values of the matching dialog object data attributes.
setDataAttribute	Sets the state of a dialog control using the value of a data attribute of the Rexx dialog.
setDataStem	Sets the *data*, (the state,) of a number of Windows dialog controls to the values specified by a stem.
setDlgFont	Sets the font that to be used for the underlying Windows dialog, when it is created.
setEditData	Sets the *data* of an edit control.
setFocus	sets the input focus to a dialog control specified by hwnd and returns the window handle of the control that previously had the focus.
setFocusToWindow	Moves the input focus to another top-level window or dialog.
setFont	Sets a new font to be used by the dialog.
setForegroundWindow	Brings the specified window to the foreground.
setGroup	Adds or removes the *group* style for the control specified.
setHScrollPos	Sets the thumb position of the horizontal scroll bar contained in the dialog.
setListBoxData	Sets the *data* of the underlying list box to the value specified.
setListColumnWidth	Sets the width of all columns in a list box in dialog units. **(Inaccurate)**
setListColumnWidthPx	Sets the width of all columns in a list box in pixels.

Dialog Method	Description
setListItemHeight	Sets the height for all items in a list box in dialog units. **(Inaccurate)**
setListItemHeightPx	Sets the height for all items in a list box in dialog units.
setListWidth	Sets the scrollable width of a list box in dialog units. **(Inaccurate)**
setListWidthPx	Sets the scrollable width of a list box in pixels.
setListTabulators	Sets the tabulators for a list box.
setRadioButtonData	Sets the *data* of a radio button control.
setRect	Moves and / or resizes the dialog.
setSBPos	Sets the current position of a scroll bar control.
setSBRange	Sets the range of a scroll bar control.
setStyleRaw	Sets the value of the dialog's style flags using the numeric value specified.
setTabStop	Add or remove the tab stop style for the specified control.
setText	Sets the text, the caption, of the dialog.
setTextAlign	Sets the text alignment option for the specified device context.
setTitle	Sets the text of the dialog.
setVScrollPos	Sets the thumb position of the vertical scroll bar contained in the dialog.
setWindowPos	Changes the size, position, visibility, and Z order of the dialog.
setWindowRect	Sets new coordinates for the specified window.
setWindowText	Sets the text for the specified window.
show	Sets the dialog window's show state.
showControl	Makes the specified dialog control reappear on the screen.
showControlFast	Shows a dialog control without redrawing its area.
showFast	Marks the dialog as visible.
showWindow	Shows the window or dialog control again.
showWindowFast	Marks the specified window as visible but does not force it to redraw.
sizeWindow	Changes the size, visibility, and Z order of the dialog.
tabToNext	Sets the focus to the next tab stop dialog control in the dialog and returns the window handle of the dialog control that currently has the focus.
tabToPrevious	Sets the focus to the previous tab stop dialog control in the dialog and returns the window handle of the dialog control that currently has the focus.
tiledBackgroundBitmap	Sets a bitmap as the dialog's background brush.
title	Gets the text of the dialog.
title=	Sets the text of the dialog.
toTheTop	Makes the dialog the topmost dialog.
transparentText	Sets the text drawing mode in a device context to transparent, (background is not changed.)
update	Invalidates the entire client area of the dialog.

Dialog Method	Description
updateWindow	Updates the client area of this window by sending a paint message to the window, if the window's update region is not empty.
validate	A method meant to be over-ridden. Its purpose is to validate the user's input and decide whether or not to allow the dialog to close.
vScrollPos	Returns the position of the vertical scroll bar in the dialog.
windowRect	Returns a **`Rect`** object containing the dimensions of the dialog in pixels.
write	Writes text in a dialog using the given font, style, and color at the position specified.
writeDirect	Writes text in a dialog at the position specified using a device context.
writeToWindow	Writes text to the dialog or dialog control, specified by its window handle, in the given font at the specified position.
writeToControl	Writes text to the dialog control, specified by its resource ID, in the given font at the specified position.

3.2. Constant Methods

The dialog object provides a number of *constant* values through the use of the **`::constant`** directive. The constants are listed and documented in this section.

Recall that the constant methods defined by the **`::constant`** directive are both class and instance methods of the class they are defined in. The constants listed here are defined in the *PlainBaseDialog* class. Therefore to access the constant value, the programmer uses either the **`PlainBaseDialog`** class object, or an instantiated dialog object. An example using the class object and the IDOK constant:

```
  dlg = .SimpleDialog~new( , "simple.h")

  ret = dlg~execute("SHOWTOP", IDI_DLG_OOREXX)
  if ret == .PlainBaseDialog~IDOK then do
    -- The user closed the dialog with Ok, do some processing ...
  end
  else do
    -- The user closed the dialog with cancel ...
  end
```

In the above example, the dialog object itself could have been used, and is probably easier to type:

```
  dlg = .SimpleDialog~new( , "simple.h")

  ret = dlg~execute("SHOWTOP", IDI_DLG_OOREXX)
  if ret == dlg~IDOK then do
    -- The user closed the dialog with Ok, do some processing ...
  end
  else do
    -- The user closed the dialog with cancel ...
  end
```

One last example using the dialog object with the SIZE_MAXIMIZED constant:

```
::method onResize unguarded
  expose u sizing minMaximized lastSizeInfo
  use arg sizingType, sizeinfo
  -- Save the size information so we know the final size of the dialog.
  lastSizeInfo = sizeInfo

  if sizingType == self~SIZE_MAXIMIZED | sizingType == self~SIZE_MINIMIZED then do
    minMaximized = .true
    if sizingType == self~SIZE_MAXIMIZED then do
      u~resize(self, sizeinfo)
      self~redrawClient(.true)
    end
  end
  else if sizingType == self~SIZE_RESTORED, minMaximized then do
    minMaximized = .false
    u~resize(self, sizeinfo)
    self~redrawClient(.true)
  end
  else do
    -- We are resizing now.
    sizing = .true
  end
```

The constants provided by the dialog object are listed in the table below:

Table 3.2. Dialog Object Constant Reference

Constant Symbol	Description
IDOK	Microsoft defined resource ID, often used for an Ok button. (Value == 1).
IDCANCEL	Microsoft defined resource ID, often used for a Cancel button. (Value == 2).
IDABORT	Microsoft defined resource ID, often used for an Abort button. (Value == 3).
IDRETRY	Microsoft defined resource ID, often used for a Retry button. (Value == 4).
IDIGNORE	Microsoft defined resource ID, often used for an Ignore button. (Value == 5).
IDYES	Microsoft defined resource ID, often used for a Yes button. (Value == 6).
IDNO	Microsoft defined resource ID, often used for a No button. (Value == 7).
IDCLOSE	Microsoft defined resource ID, often used for a Close button. (Value == 8).
IDHELP	Microsoft defined resource ID, often used for a Help button. (Value == 9).
IDTRYAGAIN	Microsoft defined resource ID, often used for a Try Again button. (Value == 10).
IDCONTINUE	Microsoft defined resource ID, often used for a Continue button. (Value == 11).
IDTIMEOUT	Microsoft defined resource ID. (Value == 32000).
SIZE_RESTORED	Used by the OS in a RESIZE event to indicate the window size was restored (Value == 0).
SIZE_MINIMIZED	Used by the OS in a RESIZE event to indicate the window was minimized (Value == 1).
SIZE_MAXIMIZED	Used by the OS in a RESIZE event to indicate the window was maximized (Value == 2).
SIZE_MAXSHOW	Used by the OS in a RESIZE event. Sent to all pop-up windows when some other window has been restored to its former size. (Value == 3).

3.3. Class Methods

3.3.1. getFontName (Class method)

```
>>--getFontName---------------------------------><
```

Returns the current default dialog font name. The default dialog font is used whenever a dialog template, used in a dynamically defined dialog, does not specify a font.

Arguments:

There are no arguments.

Return value:

This method returns the current default font family name. For instance, MS Shell Dlg.

Example:

The following example temporarily changes the default font to run the accounting program. The accounting program uses a large number of dynamically defined dialogs. It does not specify the dialog font for any of the dialogs. Before starting the program, the default font is changed to 10 pt Tahoma. Then, all the dialogs created while the accounting program is executing will be created using 10 pt Tahoma. When the accounting program is done, the old default is restored.

```
oldName = .PlainBaseDialog~getFontName
oldSize = .PlainBaseDialog~getFontSize
.PlainBaseDialog~setDefaultFont("Tahoma", 10)

ret = excuteAccounting("Daily")

.PlainBaseDialog~setDefaultFont(oldName, oldSize)
```

3.3.2. getFontSize (Class method)

```
>>--getFontSize---------------------------------><
```

Returns the current default dialog font size. The default dialog font is used whenever a dialog template, used in a dynamically defined dialog, does not specify a font.

Arguments:

There are no arguments.

Return value:

This method returns the current default font size. For instance, 8.

Example:

See the previous getFontName *example*.

3.3.3. new (Class method)

```
>>--new(--+---------+--,--id--+------------+--+---------+--)-----------------><
          +-library-+         +-,--dlgData.-+  +-,--hFile-+
```

Although the dialog object is an abstract class that the programmer will not instantiate directly, the programmer should understand the arguments of the dialog object's *new*() method. Recall that in ooRexx, when a new object is instantiated, the *new* method invokes the *init*() method of the object, using the arguments sent to the *new*() method. So, the arguments of the *new* method are also the arguments of the *init* method.

Arguments:

The arguments are:

library [optional]

The name of the file containing the dialog template. Subclasses that do not use a file containing the dialog template, such as the **`UserDialog`**, pass an empty string for this argument.

id [required]

The resource ID of the dialog within the resource file. This may be numeric or *symbolic*.

dlgData. [optional]

A dialog data stem variable (don't forget the trailing period) that contains data used to initialize the underlying dialog's controls. This is part of the automatic data *detection* feature. When *initAutoDetection* data detection is off, using this argument does nothing.

Each index of the data stem should be the resource ID of one of the dialog's controls. The value at that index is used to set the state of the corresponding, connected, dialog control when the underlying dialog is created. If the stem does not contain an index for a connected dialog control, then the state of that control is set to the empty string. The ooDialog framework keeps track of the data stem argument when it is used. If a data stem argument is used, when the dialog is closed with the *ok* command, the data of each connected dialog control is copied to the stem at the index corresponding to the dialog control's resource ID.

Note: when the argument is used the behavior of automatic data detection is changed in this way. When the underlying dialog is created, the data stem **only** is used to set the state of the dialog controls. The values of the data *attributes* are ignored. However, when the dialog closes with the *ok* command, the values of both the data attributes **and** the data stem indexes are updated.

hFile [optional]

A file, (often called a header *file*,) defining *symbolic* IDs for resources. The symbolic IDs defined within the file will be added to the *constDir* directory.

Details:

The dialog object is abstract, the programmer can not directly instantiate a workable dialog object. Rather, the programmer instantiates one of the concrete subclasses. Such as a subclass of a *RcDialog* or a *ResDialog*.

Example:

The following example shows how a header file, symbolic IDs, and the data stem can be used in instantiating a new object that is a subclass of the ResDialog. Assume **`resources.h`** is a file in the same directory as the program file and contains the following:

```
/* resources.h */
  #define IDD_BUILD_DLG           100
  #define IDC_EDITFIELD_INCLUDE   110
  #define IDC_COMBOBOX_PROJECT    120
  #define IDC_RADIO_DEBUG         130
  #define IDC_RADIO_RELEASE       131
  #define IDC_CHECKBOX_CLEAN      140
  #define IDI_DLG_ICON            514
```

The dialog (a fictitious build dialog) will have an edit control, a combo box, two radio buttons, and a check box. The **`dlgData.`** stem will be used to initialize the state of the controls using the symbolic IDs defined in the header file.

```
/* BuildDlg.rex */
  DlgData.IDC_EDITFIELD_INCLUDE = "C:\sdk\include"
  DlgData.IDC_COMBOBOX_PROJECT = "Calculator"
  DlgData.IDC_RADIO_DEBUG = 0
  DlgData.IDC_RADIO_RELEASE = 1
  DlgData.IDC_CHECKBOX_CLEAN = 0

  dlg = .BuildDialog~new("dlg.dll", IDD_BUILD_DLG, DlgData., "resources.h")
  if dlg~initCode <> 0 then do
    say "Error starting dialog.  initCode:" dlg~initCode
    return dlg~initCode
  end

  dlg~execute("NORMAL", IDI_DLG_ICON)
  ...
```

At this point the dialog is shown, the edit control will contain "C:\sdk\include", the combo box will have the Calculator project selected, the release radio button will be checked, (the debug radio button will not be checked,) and the clean check box will not be checked.

The user interacts with the dialog and selects ok to close it. Now the state of the dialog when it was closed can be determined by checking the **`dlgData.`** stem values.

```
  ...

  if DlgData.IDC_CHECKBOX_CLEAN == 1 then doClean()

  includePath = DlgData.IDC_EDITFIELD_INCLUDE
  if DlgData.IDC_RADIO_RELEASE == 1 then
    success = doReleaseBuild(includePath)
  else
    success = doDebugBuild(includePath)

  return success
```

3.3.4. setDefaultFont (Class method)

```
>>--setDefaultFont(--fontName--,--fontSize--)----><
```

This method changes the default dialog font used for all dialogs. The default dialog font is used whenever a dynamically defined dialog template does not specify a font. Since it is very unusual for a resource script to not specify a font, this mostly effects the *UserDialog* classes. Binary compiled dialog templates, subclasses of the *ResDialog* class, can not be changed, so the default dialog font has no meaning for a **`ResDialog`**.

Currently the default font is **`MS Shell Dlg`** with a size of **`8`**. MS Shell Dlg is not a true font name, but rather a pseudo name that signals the operating system to use a standard font for the specific version of Windows. In other words, on Windows 2000, MS Shell Dlg will cause the operating system to use the standard font for dialogs on Windows 2000. On XP, the operating system will use the standard XP dialog font. (The two fonts are different.) By using this font for the default, ooDialog produces dialogs that match what is most common on the current operating system.

Of course, when the ooDialog programmer specifies a font in either the *create*() or the *createCenter*() methods then the default font is ignored. Once the default font is changed in a process then all dynamic dialogs, that don't specify a font, that are created afterwards will use the new default font.

The actual font used by a dialog directly effects the value of a *dialog unit*.

Arguments:

The arguments are:

fontName

The family name to use for the default font, for example **`Tahoma`**.

fontSize

The size of the font, for example, **`10`**.

Return value:

This method does not return a value.

Example:

This example shows a change to the oostddlg.rex sample program. The default font is changed to Tahoma pt 10. This causes all the Standard *Dialogs* to be created using this font.

```
/*--------------------------------------------------------------------------*/
/*                                                                          */
/* oodialog\samples\oostddlg.rex   Standard Dialog demonstration            */
/*                                                                          */
/*--------------------------------------------------------------------------*/

.PlainBaseDialog~setDefaultFont("Tahoma", 10)
say
say 'Starting standard dialog demonstration...'

...
```

3.4. Attribute Methods

This section describes the attributes of the dialog object.

3.4.1. autoDetect (Attribute)

```
>>--autoDetect-----------------------------------><

>>--autoDetect = onOff---------------------------><
```

Reflects whether automatic data field *detection* is on or off.

autoDetect get:

When *autoDetect* is **`.true`** automatic data field detection is on. When **`.false`**, automatic detection is off.

autoDetect set:

Set *autoDetect* to **`.true`** or **`.false`** to automatic detection on or off. A syntax condition is raised if the programmer attempts to set the attribute to any other value.

Remarks:

Although the programmer *can* access the *autoDetect* attribute directly, it will not be of much use. The *initAutoDetection* method should be over-ridden to turn automatic data field detection on or off.

3.4.2. constDir (Attribute)

```
ResourceUtils::constDir


>>--constDir-------------------------------------><

>>--constDir[symbol] = numericValue--------------><
```

3.4.3. dlgHandle (Attribute)

```
>>--dlgHandle------------------------------------><

>>--dlgHandle = varName--------------------------><
```

Reflects the window *handle* of the *underlying* dialog this Rexx dialog represents.

dlgHandle get:

The *dlgHandle* is used to retrieve the window handle of this dialog for use in methods that required a handle.

dlgHandle set:

The ooDialog programmer can not set the *dlgHandle* attribute. It is set to the correct value by the ooDialog framework.

Remarks:

Dialog objects also have the *hwnd* attribute which is inherited from the *WindowBase* class. The two attributes are essentially the same and can be used by the programmer interchangeably.

Example:

This example uses the window handle of the dialog to invoke the *clearWindowRect* method:

```
  self~clearWindowRect(self~dlgHandle)
```

3.4.4. dlgID (Attribute)

```
>>--dlgID----------------------------------------><

>>--dlgID = id-----------------------------------><
```

The *dlgID* attribute reflects a whole number value that can be used to identify this dialog. This is similar to the resource *ID* of operating system resources, but it is for the use of the Rexx programmer and the ooDialog framework.

dlgID get:

Returns the dialog ID number of this dialog. This may be -1 if the dialog ID was not set. Otherwise, it will always be a non-zero positive whole number

dlgID set:

The *dlgID* can be set at any time by the programmer to a non-zero positive whole number. A *symbolic* ID can be used as long as the ooDialog framework can resolve it to a valid ID number. Any other value will not be accepted.

Remarks:

By default, the ooDialog framework will set the *dlgID* attribute to the value of the resource ID of a *ResDialog* or *RcDialog* subclass during the initialization of a dialog object. Otherwise, the *dlgID* will be set to -1. -1 signals that the ID has not been set.

The primary purpose of the *dlgID* attribute is to give the programmer a way to identify specific dialogs to the ooDialog framework. See for instance the *pagedTab* method.

The programmer is free to use this attribute for her own purposes, with the caveat that the purpose should not interfere with its use in the few methods that do rely on the *dlgID*. These methods are clearly marked so that there is no way the programmer can unknowingly interfere with the ID's use.

Although the *dlgID* gets set to the resource ID of a **`ResDialog`** or **`RcDialog`** subclass by default, changing the *dlgID* will not change the resource ID used to identify the dialog to the operating system.

Note: For **`ResDialog`** and **`RcDialog`** subclasses, *if* symbolic IDs are used to specify the resource ID in the *ResDialog* method, the *dlgID* attribute will only be set to the correct value if the ooDialog framework can resolve the symbolic ID during the *new* method. If the symbolic ID can not be resolved, the ID will be set to -1.

Take this example:

```
  dlg = .SimpleResDlg~new("simple.dll", IDD_SIMPLE_DLG)

  .application~useGlobalConstDir('O', 'simple.h')

  say 'Dialog ID:' dlg~dlgID

  dlg~execute("SHOWTOP", IDI_DLG_OOREXX)

/* Output will be: */

Dialog ID: -1
```

The *dlgID* is -1 because during the *new* method, the symbolic ID, IDD_SIMPLE_DLG, could not be resolved. The program works fine and the dialog is shown as expected because during the *execute* method, the ooDialog framework is able to resolve the symbolic ID correctly and pass its numeric value on to the operating system. Contrast the above example with the following:

```
  dlg = .SimpleResDlg~new("simple.dll", IDD_SIMPLE_DLG)

  .application~useGlobalConstDir('O', 'simple.h')

  say 'Dialog ID:' dlg~dlgID

  say 'Setting the dialog ID attribute.'
  dlg~dlgID = IDD_SIMPLE_DLG
  say 'Dialog ID:' dlg~dlgID

  dlg~execute("SHOWTOP", IDI_DLG_OOREXX)

/* Output will be: */

Dialog ID: -1
Setting the dialog ID attribute.
Dialog ID: 101
```

Details

Raises syntax errors when incorrect usage is detected.

3.4.5. factorX (Attribute)

```
WindowBase::factorX


>>--factorX--------------------------------------><

>>--factorX = ratio------------------------------><
```

3.4.6. factorY (Attribute)

```
WindowBase::factorY


>>--factorY--------------------------------------><

>>--factorY = ratio------------------------------><
```

3.4.7. fontName (Attribute)

```
>>--fontName-------------------------------------><

>>--fontName = nameVar---------------------------><
```

Prior to the creation of the underlying Windows dialog, the font name attribute specifies the name of the font that will be used to create the dialog. This font is used in a *UserDialog* when the font is not specified in the *create*() or the *createCenter*() methods, or in a *RcDialog* where the font is not specified in the resource script file. With a binary compiled dialog resource, subclasses of the *ResDialog* class, the font has already been specified when the dialog template was compiled and can not be changed. Therefore the font name attribute has no effect on a **`ResDialog`** prior to the execution of the dialog.

After the creation of the dialog, the attribute always reflects the font name that was actually used the underlying dialog was created. The public *setDlgFont* method can be used to set both the font name and font size attributes at the same time.

fontName get:[public]

Returns the name of the font as a string.

fontName set:[private]

Sets the font name to that specified. The name must be less than 256 characters long.

Remarks:

In order to accurately calculate *dialog unit* in methods such as *getTextSizeDu*(), it is important that the *fontName* and *fontSize* attributes match the actual font that the dialog will use. Therefore, whenever dialog unit calculations are going to be made before the underlying dialog is created, it is important to set these attributes to match the font that will be used in the dialog. This must be done before the dialog unit calculations.

Note that the font name and font size attributes are set in the super class init() method to the value of the *setDefaultFont* dialog font. Therefore, to have any effect, setting the font name attribute has to be done after the super class init() is finished.

Details:

The *fontName* attribute is a member of the *PlainBaseDialog* class.

Syntax errors are raised when incorrect usage is detected.

Example:

This example is a portion of the code used to to create a message box dialog using a variable font. The calcSizes() method is not shown, but in that method the size and position of the controls, and the overall size of the dialog, are calculated in relation to the size needed for the message.

```
  fontName = "Tahoma"
  fontSize = 20
  message = "Drive z: is a network drive and is not accessible."
  title = "Disk Drive Error"

  dlg = .MessageBox~new( , , message, title, fontName, fontSize)
  if dlg~initCode = 0 then do
    dlg~execute("SHOWTOP", 14)
  end

return 0
-- End of entry point.

::requires "ooDialog.cls"

::class 'MessageBox' subclass UserDialog

::method init
  expose cx cy message title fontName fontSize

  a = .array~new(2)
  if arg(1, 'E') then a[1] = arg(1)
  if arg(2, 'E') then a[2] = arg(2)

  message = arg(3)
  title = arg(4)
  if arg(5, 'E') then do
    fontName = arg(5)
    fontSize = arg(6)
  end

  forward class (super) arguments (a) continue
  if self~initCode <> 0 then return

  if arg(5, 'E') then do
    self~fontName = fontName
    self~fontSize = fontSize
  end

  self~calcSizes()
```

```
    self~createCenter(cx, cy, title)
```

3.4.8. fontSize (Attribute)

```
>>--fontSize-------------------------------------><

>>--fontSize = sizeVar---------------------------><
```

The *fontSize* attribute is the counterpart to the *fontName* attribute. The description and remarks for the *fontName* attribute apply equally well to the *fontSize* attribute.

fontSize get:[public]

Returns the point size of the font.

fontSize set:[private]

Sets the point size of the font name to the non-negative number specified. Although 0 is accepted, setting the point size to 0 is not advised.

Remarks:

See the remarks for the *fontName* attribute.

Details:

The *fontSize* attribute is a member of the *PlainBaseDialog* class.

Syntax errors are raised when incorrect usage is detected.

Example:

See the fontName *example*/>, which also uses the fontSize attribute.

3.4.9. hwnd (Attribute)

```
WindowBase::hwnd


>>--hwnd-----------------------------------------><
```

3.4.10. initCode (Attribute)

```
WindowBase::initCode


>>--initCode-------------------------------------><

>>--initCode = code------------------------------><
```

3.4.11. ownerDialog (Attribute)

```
>>--ownerDialog----------------------------------><
```

```
>>--ownerDialog = rexxDlg------------------------><
```

The *ownerDialog* attribute reflects the dialog that owns this dialog. An owner window and the windows it owns is a Windows operating system concept. The operating system places several constraints on owned windows. See the remarks section for a more detailed discussion of the owner window.

In the ooDialog framework, *all* *Control* dialogs *must* have an owner dialog. Dialogs like the *PropertySheetDialog* can *not* have an owner dialog. For the most part other dialogs would not have an owner dialog.

ownerDialog get:

Returns the Rexx owner dialog of this dialog, or **`.nil`** if this dialog does not have an owner dialog.

ownerDialog set:

Sets the Rexx owner dialog for this dialog. There are a number of constraints imposed on setting the owner dialog. The owner dialog can not be a *ControlDialog*, a *PropertySheetDialog*, or a *PropertySheetPage* dialog.

Once set, the owner dialog can not be changed or removed. The owner dialog can not be set once the *underlying* dialog for this dialog has been created and when this owned dialog is executed, the underlying owner dialog must have already been created.

Remarks:

Owned windows have a number of features that are imposed by the operating system. An owned window is always above its owner. This means that when the user brings the owner window to the foreground to work with it, the owned window is always completely visible. The system automatically closes an owned window when its owner window is closed. When the owner window is minimized, the owned window is hidden by the system. The system does not show owned windows on the task bar or in the alt-tab window dialog.

The operating system constraints can make owned windows useful for certain applications. A tool palette immediately comes to mind. Another application might be a form window, which would allow the user to refer back to the main window while filling out the form without *losing* the form under other windows on the desktop.

Details:

Raises syntax errors when incorrect usage in setting the attribute is detected.

Example:

This example is from an application that creates a tool palette. The tool palette dialog is owned by the main dialog. A tool palette is an ideal use of an owner dialog. The operating system ensures that the tool palette is always visible when the owner dialog is visible, and always on top of it. When the owner dialog is closed or hidden the operating system closes or hides the tool palette. The code below shows the proper sequence of assigning the owner and executing the dialogs:

```
.application~useGlobalConstDir("O", "resources\useTools.h")

dlg = .MainDialog~new

dlgTool = .ToolPaletteDlg~new
dlgTool~ownerDialog = dlg

-- Start the main dialog asynchronously so we continue and then can start the
-- tool palette.
dlg~executeAsync("SHOWTOP", IDI_DLG_OOREXX)

-- Start the tool palette now.  It can not be started until its owner dialog
-- has been started.
```

```
    dlgTool~popup("SHOWTOP")

    dlg~endAsyncExecution
```

3.4.12. pixelCX (Attribute)

```
WindowBase::pixelCX


>>--pixelCX--------------------------------------><
```

3.4.13. pixelCY (Attribute)

```
WindowBase::pixelCY


>>--pixelCY--------------------------------------><
```

3.4.14. sizeX (Attribute)

```
WindowBase::sizeX


>>--sizeX----------------------------------------><

>>--sizeX = dialogUnits--------------------------><
```

3.4.15. sizeY (Attribute)

```
WindowBase::sizeY


>>--sizeY----------------------------------------><

>>--sizeY = dialogUnits--------------------------><
```

3.5. Basic Window Methods

Recall that in the Windows graphical user interface, everything is a window. The *WindowBase* class is a mixin class that contains methods common to all windows. Since a dialog is a window, it inherits the *WindowBase* class. This section lists all the basic window methods for the dialog object.

3.5.1. childWindowFromPoint

```
WindowBase::childWindowFromPoint


>>--childWindowFromPoint(--point--+----------+--)------------------------------><
                                  +-,-flags--+
```

3.5.2. clear

```
WindowBase::clear


>>--clear----------------------------------------><
```

3.5.3. client2screen

```
WindowBase::client2screen


>>--client2screen(--pointOrRect--)---------------><
```

3.5.4. clientRect

```
WindowBase::clientRect


>>--clientRect(--+--------+--)-------------------><
                 +--hwnd--+
```

3.5.5. clientToScreen

```
WindowBase::clientToScreen


>>--clientToScreen(--x--,--y--)------------------><
```

3.5.6. disable

```
WindowBase::disable


>>--disable--------------------------------------><
```

3.5.7. display

```
WindowBase::display
```

3.5.8. draw

```
WindowBase::draw


>>--draw-----------------------------------------><
```

3.5.9. enable

```
WindowBase::enable


>>--enable---------------------------------------><
```

3.5.10. foregroundWindow

```
WindowBase::foregroundWindow


>>--foregroundWindow-----------------------------><
```

3.5.11. getClientRect

```
WindowBase::getClientRect


>>--getClientRect(--+------+--)------------------><
                    +-hwnd-+
```

3.5.12. getExStyleRaw

```
WindowBase::getExStyleRaw


>>--getExStyleRaw--------------------------------><
```

3.5.13. getID

```
WindowBase::getID


>>--getID----------------------------------------><
```

3.5.14. getPos

```
WindowBase::getPos


>>--getPos---------------------------------------><
```

3.5.15. getRealPos

```
WindowBase::getSize


>>--getRealPos-----------------------------------><
```

3.5.16. getRealSize

```
WindowBase::getRealSize


>>--getRealSize----------------------------------><
```

3.5.17. getRect

```
WindowBase::getRect


>>--getRect--------------------------------------><
```

3.5.18. getSize

```
WindowBase::getSize


>>--getSize--------------------------------------><
```

3.5.19. getStyleRaw

```
WindowBase::getStyleRaw


>>--getStyleRaw----------------------------------><
```

3.5.20. getText

```
WindowBase::getText


>>--getText--------------------------------------><
```

3.5.21. getTextSizePx

```
WindowBase::getTextSizePx


>>--getTextSizePx(-text--)-----------------------><
```

3.5.22. getTextSizeScreen

```
WindowBase::getTextSizeScreen


>>--getTextSizeScreen(-text--+---------+--+------------+--+-------------+-)----><
                             +-,-type--+  +-,-fontSrc--+  +-,-fontSize--+
```

3.5.23. hide

```
WindowBase::hide


>>--hide-----------------------------------------><
```

3.5.24. hideFast

```
WindowBase::hideFast


>>--hideFast-------------------------------------><
```

3.5.25. isEnabled

```
WindowBase::isEnabled


>>--isEnabled------------------------------------><
```

3.5.26. isVisible

```
WindowBase::isVisible


>>--isVisible------------------------------------><
```

3.5.27. mapWindowPoints

```
WindowBase::mapWindowPoints


>>--mapWindowPoints(--hwndTo--,--points--)-------><
```

3.5.28. move

```
WindowBase::move


>>--move(--xPos--,--yPos--+------------+--)-----><
                          +-,-showOpts--+
```

3.5.29. moveTo

```
WindowBase::moveTo


Form 1:

>>--moveTo(--point--+--------------+--)----------><
```

```
                    +--,-showOpts--+
Form 2:

>>--moveTo(--x,--y--+--------------+--)---------><
                    +--,-showOpts--+
Generic form:

>>--moveTo(--newPos--+--------------+--)--------><
                     +--,-showOpts--+
```

3.5.30. moveWindow

```
WindowBase::moveWindow


Form 1:

>>--moveWindow(--hwndBehind--,--point--+--------------+--)------><
                                       +--,-showOpts--+
Form 2:

>>--moveWindow(--hwndBehind--,--x,--y--+--------------+--)------><
                                       +--,-showOpts--+
Generic form:

>>--moveWindow(--hwndBehind--,--newPos--+--------------+--)-----><
                                        +--,-showOpts--+
```

3.5.31. redraw

```
WindowBase::redraw


>>--redraw---------------------------------------><
```

3.5.32. redrawClient

```
WindowBase::redrawClient


>>--redrawClient(--+------------+--)------------><
                   +--eraseBkg--+
```

3.5.33. resize

```
WindowBase::resize


>>--resize(--width--,--height--+-------------+--)---------------><
                               +-,-showOpts--+
```

3.5.34. resizeTo

```
WindowBase::resizeTo
```

```
Form 1:

>>--resizeTo(--size--)---------------------------><

Form 2:

>>--resizeTo(--cx,--cy--)------------------------><

Generic form:

>>--resizeTo(--newSize--)------------------------><
```

3.5.35. screen2client

```
WindowBase::screen2client


>>--screen2client(--pointOrRect--)---------------><
```

3.5.36. screenToClient

```
WindowBase::screenToClient


>>--screenToClient(--x--,--y--)------------------><
```

3.5.37. sendMessage

```
WindowBase::sendMessage


>>--sendMessage(--id--,--msg--,--wParam--,--lParam--)-----------><
```

3.5.38. sendMessageHandle

```
WindowBase::sendMessageHandle


>>--sendMessageHandle(--id--,--msg--,--wParam--,--lParam--)-----><
```

3.5.39. setRect

```
WindowBase::setRect


Form 1:

>>--setRect(--rectangle--+------------+--)-------><
                         +-,-showOpts-+

Form 2:

>>--setRect(--point--,--size--+------------+--)-----------------><
                              +-,-showOpts-+
Form 3:

>>--setRect(--x-,--y-,--cx-,--cy--+------------+--)-------------><
                                  +-,-showOpts-+

Generic form:

>>--setRect(--ptSizeRectangle--+------------+--)----------------><
                               +-,-showOpts-+
```

3.5.40. setStyleRaw

```
WindowBase::setStyleRaw


>>--setStyleRaw(--value--)-----------------------><
```

3.5.41. setText

```
WindowBase::setText


>>--setText(--newText--)-------------------------><
```

3.5.42. setTitle

```
WindowBase::setTitle


>>--setTitle(--newText--)------------------------><
```

3.5.43. setWindowPos

```
WindowBase::setWindowPos


Form 1:

>>--setWindowPos(--hwndBehind--,--rectangle--+------------+--)-----------------><
                                             +-,-showOpts-+

Form 2:

>>--setWindowPos(--hwndBehind--,--point--,--size--+------------+--)------------><
                                                  +-,-showOpts-+

Form 3:

>>--setWindowPos(--hwndBehind--,--x-,--y-,--cx-,--cy--+------------+--)--------><
                                                      +-,-showOpts-+

Generic form:

>>--setWindowPos(--hwndBehind--,--ptSizeRectangle--+------------+--)-----------><
                                                   +-,-showOpts-+
```

3.5.44. showFast

```
WindowBase::showFast


>>--showFast-------------------------------------><
```

3.5.45. sizeWindow

```
WindowBase::sizeWindow


Form 1:

>>--sizeWindow(--hwndBehind--,--size--+-------------+--)-------><
                                      +--,-showOpts--+

Form 2:

>>--sizeWindow(--hwndBehind--,--cx,--cy--+-------------+--)----><
                                         +--,-showOpts--+

Generic form:

>>--sizeWindow(--hwndBehind--,--newSize--+-------------+--)----><
                                         +--,-showOpts--+
```

3.5.46. title

```
WindowBase::title


>>--title------------------------------------------><
```

3.5.47. title=

```
WindowBase::title=


>>--title=newText--------------------------------><
```

3.5.48. update

```
WindowBase::update


>>--update---------------------------------------><
```

3.5.49. updateWindow

```
WindowBase::updateWindow
```

```
>>--updateWindow---------------------------------><
```

3.5.50. windowRect

```
WindowBase::windowRect


>>--windowRect(--+--------+--)-------------------><
                 +--hwnd--+
```

3.6. Extended Window Methods

The methods implemented by *WindowsExtensions* class are listed in this section. The class name, *WindowExtensions* would seem to imply that the methods were common to all windows. However, the methods of this class are really just extensions to the original ooDialog framework, and many of the methods are not window specific methods.

3.6.1. createBrush

```
WindowExtensions::createBrush


>>--createBrush(--+---------+--+-----------------+--)------------------------><
                  +--color--+  +-,-brushSpecifier-+
```

3.6.2. createFont

```
WindowExtensions::createFont


>>--createFont(--+----------+-+------------+-+---------+-+-------------+--)----><
                 +-fontName-+ +-,-fontSize-+ +-,-style-+ +-,-fontWidth-+
```

3.6.3. createFontEx

```
WindowExtensions::createFontEx


>>--createFontEx(--fontName-+--------------+--+---------------+--)-------------><
                            +-,-pointSize--+  +-,-additional--+
```

3.6.4. createPen

```
WindowExtensions::createPen


>>--createPen(--+-------+--+----------+--+---------+--)-----------------------><
                +-width-+  +-,-style--+  +-,-color--+
```

3.6.5. deleteFont

```
WindowExtensions::deleteFont


>>--deleteFont(--hFont--)----------------------------><
```

3.6.6. deleteObject

```
WindowExtensions::deleteObject


>>--deleteObject(--obj--)----------------------------><
```

3.6.7. drawAngleArc

```
WindowExtensions::drawAngleArc


>>--drawAngleArc(-dc-,-xs-,-ys-,-x-,-y-,-radius-,-startangle-,-sweepangle-)----><
```

3.6.8. drawArc

```
WindowExtensions::drawArc


>>--drawArc(--dc-,-x-,-y-,-x2-,-y2--+-------+--+-------+--+-------+--+-------+--)---><
                                    +-,-r1x-+  +-,-r1y-+  +-,-r2x-+  +-,-r2y-+
```

3.6.9. drawLine

```
WindowExtensions::drawLine


>>--drawLine(--dc--,--+-------+--,--+-------+--,--toX--,--toY--)-><
                      +-fromX-+     +-fromY-+
```

3.6.10. drawPie

```
WindowExtensions::drawPie


>>--drawPie(--dc-,-x-,-y-,-x2-,-y2--+-------+--+-------+--+-------+--+-------+--)---><
                                    +-,-r1x-+  +-,-r1y-+  +-,-r2x-+  +-,-r2y-+
```

3.6.11. drawPixel

```
WindowExtensions::drawPixel
```

```
>>--drawPixel(--dc--,--x--,--y--,--color--)----------><
```

3.6.12. fillDrawing

```
WindowExtensions::fillDrawing

>>--fillDrawing(--dc--,--x--,--y--,--color--)----><
```

3.6.13. fillRect

```
WindowExtensions::fillRect

>>--fillRect(--dc--,--rect--,--brush--)----------><
```

3.6.14. fontColor

```
WindowExtensions::fontColor

>>--fontColor(--color--,--dc--)----------------------><
```

3.6.15. fontToDC

```
WindowExtensions::fontToDC

>>--fontToDC(--dc--,--hFont--)-----------------------><
```

3.6.16. freeDC

```
WindowExtensions::freeDC

>>--freeDC(--dc--)-----------------------------------><
```

3.6.17. getArcDirection

```
WindowExtensions::getArcDirection

>>--getArcDirection(--dc--)--------------------------><
```

3.6.18. getDC

```
WindowExtensions::getDC
```

```
>>--getDC--------------------------------------------><
```

3.6.19. getFont

```
WindowExtensions::getFont

>>--getFont------------------------------------------------------><
```

3.6.20. getPixel

```
WindowExtensions::getPixel

>>--getPixel(--dc--,--x--,--y--)---------------------><
```

3.6.21. getSysBrush

```
WindowExtensions::getSysBrush

>>--getSysBrush(--sysColor--)--------------------><
```

3.6.22. getTextAlign

```
WindowExtensions::getTextAlign

>>--getTextAlign(--hDC--)-----------------------><
```

3.6.23. getTextExtent

```
WindowExtensions::getTextExtent

>>--getTextExtent(--hDC--,--text--)--------------><
```

3.6.24. hScrollPos

```
WindowExtensions::hScrollPos

>>--hScrollPos---------------------------------------><
```

3.6.25. loadBitmap

```
WindowExtensions::loadBitmap

>>--loadBitmap(--bmpFilename--+------------+--)------><
```

```
                                   +-,-loadOpt--+
```

3.6.26. objectToDC

```
WindowExtensions::objectToDC


>>--objectToDC(--dc--,--obj--)-----------------------><
```

3.6.27. opaqueText

```
WindowExtensions::opaqueText


>>--opaqueText(--dc--)-------------------------------><
```

3.6.28. rectangle

```
WindowExtensions::rectangle


>>--rectangle(--dc--,--x--,--y--,--x2--,--y2--+-----------+--)-----------------><
                                              +-,-keyWord-+
```

3.6.29. removeBitmap

```
WindowExtensions::removeBitmap


>>--removeBitmap(--hBitmap--)------------------------><
```

3.6.30. scroll

```
WindowExtensions::scroll


>>--scroll(--cx--,--cy--)----------------------------><
```

3.6.31. setArcDirection

```
WindowExtensions::setArcDirection


>>--setArcDirection(--dc--+--------------+--)--------><
                          +-,-direction--+
```

3.6.32. setFont

```
WindowExtensions::setFont
```

```
>>--setFont(--fontHandle--+---------+--)------------------------><
                          +-,redraw-+
```

3.6.33. setHScrollPos

```
WindowExtensions::setHScrollPos

>>--setHScrollPos(--position--+-----------+--)-------><
                              +-,-redraw--+
```

3.6.34. setTextAlign

```
WindowExtensions::setTextAlign

>>--setTextAlign(--hDC--,--align--)--------------><
```

3.6.35. setVScrollPos

```
WindowExtensions::setVScrollPos

>>--setVScrollPos(--position--+-----------+--)-------><
                              +-,-redraw--+
```

3.6.36. transparentText

```
WindowExtensions::transparentText

>>--transparentText(--dc--)-------------------------><
```

3.6.37. vScrollPos

```
WindowExtensions::vScrollPos

>>--vScrollPos---------------------------------------><
```

3.6.38. write

```
WindowExtensions::write

>>--write(-x-,-y-,-text-+---------+-+---------+-+--------+-+------+-+------+-)---><
                        +-,-fName-+ +-,-fSize-+ +-,-opts-+ +-,-fg-+ +-,-bk-+
```

3.6.39. writeDirect

```
WindowExtensions::writeDirect


>>--writeDirect(--dc--,--xPos--,--yPos--,--text--)---><
```

3.7. Preparing and Running the Dialog

This section documents the methods used to prepare and initialize a dialog, show it, execute it, and stop it.

3.7.1. addNewAttribute

```
>>--addNewAttribute(--atrName--+-----------+--+-------------+--+-----------+--)--><
                               +-,-initVal--+  +-,-unguarded--+  +-,-private--+
```

Adds a new attribute (a new method) to this dialog.

Arguments:

The arguments are:

atrName [required]

The name of the new attribute. Must be a valid Rexx method name.

initVal [optional]

An initial value to set the attribute to. The default is the empty string.

unguarded [optional]

True or false, should the attribute be unguarded. The default is false.

private [optional]

True or false, should the attribute be private. The default is false.

Return value:

There is no return from this method.

Remarks:

This is a convenience method. It uses the **`Object`** class's *setMethod* to add the attribute to the dialog object. *setMethod* is private, but *addNewAttribute* is public. This allows *addNewAttribute* to be used to dynamically add an attribute to a dialog where only a reference to the dialog object is available. In addition, it is a convenience to already have the code to add the attribute written.

Example:

This example shows how it is useful to be able to dynamically add an attribute to a dialog object at run time. The owner dialog could be any generic dialog:

```
::method initUpdateListView private
  use strict arg lvID, pause = .005

  self~addNewAttribute('updateListViewID', lvID, .true)

  owner = self~ownerDialog
```

```
owner~addNewAttribute(updateListViewPageDialog, self, .true, .true)
owner~addNewAttribute(updateListViewLastFocused, 0, .true, .true)
```

3.7.2. addNewMethod

```
>>--addNewMethod(--mthName--+--------------+--+------------+--)----------------><
                            +-,-unguarded--+  +-,-private--+
```

Adds a new method to this dialog.

Arguments:

The arguments are:

mthName [required]

The name of the new method. Must be a valid Rexx method name.

unguarded [optional]

True or false, should the method be unguarded. The default is false.

private [optional]

True or false, should the method be private. The default is false.

Return value:

This method always returns 0.

Remarks:

This is a convenience method. It uses the **`Object`** class's *setMethod* to add the new method to the dialog object. *setMethod* is private, but *addNewMethod* is public. This allows *addNewMethod* to be used to dynamically add a method to a dialog where only a reference to the dialog object is available. In addition, it is a convenience to already have much of the code to add the method written.

Example:

This example connects an event handler to a dialog and adds the event handling method to the dialog dynamically. Note that only a reference to the dialog object is available, so it is not possible to add the method directly using the **`Object`** class's *setMethod* method.

```
::method initUpdateListView private
  use strict arg lvID, pause = .005

  ...

  owner = self~ownerDialog

  src = .array~new
  src[1]  = "expose updateListViewPageDialog updateListViewLastFocused"
  src[2]  = "use arg flag, hwnd, hFocused, isMinimized"
  src[3]  = ""
  src[4]  = "reply .false"
  src[5]  = ""
  src[6]  = "if flag == INACTIVE then updateListViewLastFocused = hFocused"
  src[7]  = "else updateListViewPageDialog~updateListView(updateListViewLastFocused)"

  owner~addNewMethod('onUpdateListViewActivate', src, .true)

  owner~connectActivate('onUpdateListViewActivate')
```

3.7.3. addAutoStartMethod

```
DialogExtensions::addAutoStartMethod


>>--addAutoStartMethod(--+---------+--,--methodName--+---------------+--)------><
                         +-inClass-+                 +-,--parameters-+
```

3.7.4. endAsyncExecution

```
DialogExtensions::endAsyncExecution


>>--endAsyncExecution----------------------------><
```

3.7.5. execute

```
>>--execute(--+-------------+--+---------+--)----><
              +--showOption-+  +-,--icon-+
```

The *execute* method, creates the *underlying* dialog, *show* it, starts the *addAutoStartMethod* methods, if any, and destroys the underlying dialog when the user closes it. If auto *initAutoDetection* is on, the dialog *data* is passed to the underlying dialog before execution and received from it after the dialog is terminated.

Arguments:

The arguments are:

showOption [optional]

Zero or one of the following keywords to specify how the dialog is shown. This keyword is used in the automatic invocation of the *show* method. Case is not significant. If this argument is omitted, the NORMAL keyword is used:

NORMAL
: Makes the dialog visible in its default position and window size. This has the effect of restoring the dialog size and position if it is minimized or maximized. This is the default if the argument is omitted.

DEFAULT
: DEFAULT is an alias for NORMAL. The two keywords are functionally identical.

SHOWTOP
: Makes the dialog visible and the topmost window.

HIDE
: Makes the dialog invisible.

MIN
: Minimizes the dialog and activates the next window in the window order.

MAX
: Maximizes, and makes visible if necessary, the dialog.

INACTIVE
: Makes the dialog visible without changing the active window. When the NORMAL keyword is used, the dialog is shown and becomes the active window. The INACTIVE keyword makes the dialog visible without changing the focus from the current active window.

RESTORE
: Makes the dialog visible and restores it to its original size and position if it was minimized or maximized. An application should specify this flag when restoring a minimized window.

icon [optional]
: The resource ID of the *dialog icon*. May be numeric or *symbolic*. If an icon ID is not supplied, the default ooDialog icon will be used.

Return value:

The return value is one of the following codes:

0
: Some error occurred, the dialog was not executed.

1
: The user terminated the dialog using an ok command.

2
: The user terminated the dialog using a cancel command.

Remarks:

Although any of the show keywords can be used, for the *execute* method the SHOWTOP, HIDE, and MAX keywords make the most sense. SHOWTOP is the usual keyword. Rather than use the HIDE keyword, it is more practical to simply create the dialog **without** the VISIBLE style.

If another ooDialog dialog has been started by the Rexx program, it is disabled when *execute* is invoked. This in effect makes the dialog a *modal* dialog. To start a *modeless* dialog use the *popup* or *popupAsChild* methods.

Example:

The following example instantiates a new *UserDialog* subclass object. The dialog *template* is started by the *create* method and finished in the *defineDialog* method. Then the *execute* method runs the dialog as the topmost window. The *dialog icon* is set to one of the pre-defined icons supplied by ooDialog.

When the user terminates the dialog, the return from *execute* is checked to determine how the user closed the dialog. Typically, if the user canceled the dialog, any changes in the dialog are ignored:

```
dlg = .MyDialog~new(...)
dlg~create(...)

ret = dlg~execute("SHOWTOP", IDI_DLG_OOREXX)
if ret == 1 then do
  -- The user closed the dialog with the ok command.
  ...
end
else do
  -- The user canceled the dialog.
  ...
end

::class 'MyDialog' subclass UserDialog
```

```
...
::method defineDialog

  self~createStaticText(...)
  self~createEdit(...)
  self~createPushButton(...)
  ...
```

3.7.6. executeAsync

```
DialogExtensions::executeAsync


>>--executeAsync(--+-----------+--+---------------+--+---------+--)------------><
                   +--ignored--+  +-,--showOption-+  +-,--icon-+
```

3.7.7. initDialog

```
>>--initDialog-----------------------------------><
```

The *initDialog* method is invoked automatically by the ooDialog framework. It is intended to be over-ridden by the programmer in her dialog subclasses. The method is used to initialize the dialog controls, if needed, after the *underlying* Windows dialog is created.

Arguments:

The ooDialog framework does not pass any arguments when invoking this method.

Return value:

Any return from this method is ignored by the ooDialog framework.

Remarks:

The ooDialog framework invokes the *initDialog* at the proper point in time, immediately after the underlying Windows dialog is created. The programmer should never invoke the method himself.

Example:

This example show how the *initDialog* method is used to initialize a list *ListBox* by adding items to it.

```
::class 'MyDialog' subclass RcDialog
::method initDialog

  lb = self~newListBox(IDC_LISTBOX)

  lb~add("this is the first line")
  lb~add("and this one the second")
```

3.7.8. isDialogActive

```
>>--isDialogActive-------------------------------><
```

The isDialogActive method returns 1 if the Windows dialog still exists.

Example:

The following example tests whether the dialog is active:

```
if MyDialog~isDialogActive then ...
```

3.7.9. parseIncludefile

```
ResourceUtils::parseIncludeFile


>>--parseIncludeFile(--fileName--)---------------><
```

3.7.10. popup

```
DialogExtensions::popup


>>--popup(--+--------------+--+------------+--+---------+--)-------------------><
            +--showOption--+  +-,-ignored--+  +-,-icon--+
```

3.7.11. popupAsChild

```
DialogExtensions::popupAsChild


>>--popupAsChild(--+-parent-+---------------+--+------------+--+---------+--)--><
                            +-,-showOption--+  +-,-ignored--+  +-,-icon--+
```

3.7.12. resolveNumericID

```
ResourceUtils::resolveNumericID


>>--resolveNumericID(--id--)---------------------><
```

3.7.13. resolveSymbolicID

```
ResourceUtils::resolveSymbolicID


>>--resolveSymbolicID(--id--)--------------------><
```

3.7.14. setDlgFont

```
>>--setDlgFont(--fontName--+-------------+--)----><
                           +-,-fontSize--+
```

Sets the font that to be used for the underlying Windows dialog, when it is created. This method sets the *fontName* and *fontSize* attributes.

Arguments:

The arguments are:

fontName

Required. The name of the font, such as Tahoma. The font name must be less than 256 characters in length.

fontSize

Optional. The point size of the font, such as 10. The default point size when this argument is omitted is 8.

Return value:

This method always returns 0.

Remarks:

The *setDlgFont* is primarily of use in a *UserDialog* or a subclasses of a *UserDialog*.

In a *ResDialog*, the font of the compiled binary resource will always be used, and the font set by this method has no effect. In a *RcDialog*, if the resource script file specifies the font, that font will be used. Likewise, in a *UserDialog*, if the programmer specifies a font in the *create*() method call, (or the *createCenter*() method call,) the specified font over-rides what is set by this method.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

This example sets the dialog font for a *UserDialog* dialog before the dialog *template* is started. This allows the code to *inaccurate* calculate the size of the bitmap in *dialog unit*s. The size of the bitmap is then used to size the dialog and to place the other controls in the dialog.

```
::class 'BanditDlg' subclass UserDialog
...

::method init
   expose kind3 initialSpeed minSpeed maxSpeed
   use arg kind3, initialSpeed, minSpeed, maxSpeed

   self~init:super()

   -- Set the font the dialog will use when created.  Without this step, dialog
   -- units can not be calculated correctly.
   self~setDlgFont('Arial', 18)

   -- Calculate the size of a bitmap in dialog units.
   bitMapSize = .Size~new(152, 178)
   self~pixel2dlgUnit(bitMapSize)

   -- Calculate the size of this dialog based on the bitmap size.
   dlgSize = self~calcSize(bitMapSize)

   ...
```

3.8. Instantiating Dialog Controls

The methods in this section are used to instantiate one of the concrete dialog *control* objects. The dialog control objects can not be instantiated directly by the programmer in Rexx code. Dialog controls are instantiated indirectly through the dialog object.

All dialog control objects are obtained through one of the methods in this section. The methods all have the same format:

```
controlObject = dialogObject~new[ControlName](resourceID)
```

where *[ControlName]* is the name Microsoft uses for the control. *resourceID* is the *resource ID* assigned to the dialog control. Dialog control objects can only be instantiated after the *underlying* dialog has been created. For any error, all the methods return the *.nil* object.

3.8.1. createToolTip

```
>>--createToolTip(--id--+----------+--)---------><
                        +-,-style--+
```

Creates the underlying Windows ToolTip control and returns an instantiated Rexx *ToolTip* object.

Arguments:

The arguments are:

id [required]

The resource ID of the ToolTip dialog control. May be numeric or *symbolic*.

style [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. If this argument is omitted, the default style is set to NOPREFIX ALWAYSTIP:

ALWAYSTIP	NOANIMATE	USEVISUALSTYLE
BALLOON	NOFADE	
CLOSE	NOPREFIX	

ALWAYSTIP

Indicates that the ToolTip control appears when the cursor is on a tool, even if the ToolTip control's owner window is inactive. Without this style, the ToolTip appears only when the tool's owner window is active.

BALLOON

Indicates that the ToolTip control has the appearance of a cartoon *balloon*, with rounded corners and a stem pointing to the item.

CLOSE

Displays a Close button on the ToolTip.

NOANIMATE

Disables sliding ToolTip animation on Microsoft Windows 2000 systems.

NOFADE

Disables fading ToolTip animation on Windows 2000 systems.

NOPREFIX

Prevents the system from stripping the ampersand character from a string. Without this style, the system automatically strips ampersand characters. This allows an application to use the same string as both a menu item and as text in a ToolTip control.

USEVISUALSTYLE

Uses themed hyperlinks. The theme will define the styles for any links in the tooltip. This style always requires *PARSELINKS* to be set.

Return value:

Returns an instantiated Rexx *ToolTip* object that represents the newly created Windows tool tip. On error, the **`.nil`** object is returned.

Remarks:

Unlike most other dialog controls, the ToolTip control can not be added to the dialog *template*. The *createToolTip* method creates the underlying ToolTip control, and must be invoked *after* the Windows dialog has been created. There is no *UserDialog* method to create a tool tip in the *defineDialog* method.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable. In some cases, the failure to create the underlying ToolTip control by the operating system can be detected in the ooDialog framework. This will result in the **`.SystemErrorCode`** being set. However, this type of failure is unlikely to happen.

Example:

This example creates a ToolTip with the *BALLOON* style and a Close button. It then gives the ToolTip a title and an icon:

```
::method initDialog
  expose icon

  toolTip = self~createToolTip(IDC_TT_BALLOON, 'BALLOON CLOSE')
  toolTip~setTitle("Important Message", icon)
  ...
```

3.8.2. newCheckBox

```
>>--newCheckBox(--id--)--------------------------><
```

The *newCheckBox* method returns an object of the *CheckBox* control class. This object represents the check box control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]

The resource ID of the check box dialog control. May be numeric or *symbolic*.

Return value:

An object of the check box control class or *.nil* on any error.

Example:

The following example determines the check state of a *3STATE* check box and displays a message announcing the state:

```
::class 'MyDlgClass' subclass ResDialog

::method currentState
```

```
  checkBox = self~newCheckBox(304)
  if checkBox == .nil then return .false
  if checkBox~checked then say "The check box is checked!"
  else if checkBox~isIndeterminate then say "The check box is in the indeterminate
 state!"
  else say "The check box is not checked!"
  return .true
```

3.8.3. newComboBox

```
>>--newComboBox(--id--)--------------------------><
```

The *newComboBox* method returns an object of the *ComboBox* control class. This object represents the combo box control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]

The resource ID of the combo box dialog control. May be numeric or *symbolic*.

Return value:

An object of the combo box control class or *.nil* on any error.

Example:

The following example comes from a fictious accounting program. In one of the dialogs for the program, when the user selects a specific city, say San Diego, the zip code combo box is populated with the valid zip codes for that city. In the program, symbolic ID *symbolic*s have been used for the controls. The valid zip codes are passed into the method as an array.

```
::class "BillingDlg" subclass RcDialog

...

::method setZipCodes
  use strict arg codes

  combo = self~newComboBox(IDC_COMBO_ZIP)
  if combo == .nil then return .false

  lowest = 99999
  combo~deleteAll
  do zipCode over codes
    combo~add(zipCode)
    if zipCode < lowest then do
      lowest = zipCode
    end
  end
  combo~select(lowest)
  return .true
```

3.8.4. newDateTimePicker

```
>>--newDateTimePicker(--id--)--------------------><
```

The *newDateTimePicker* method returns a date and time picker object that represents the date and time picker control with the specified resource ID in the underlying dialog. See the *DateTimePicker* class for details about date and time picker controls.

Arguments:

The arguments are:

id [required]

The *resource ID* of the date and time picker. This may be *symbolic* or numeric.

Return value:

A date and time picker object on success, or *.nil*.nil if the method fails for any reason.

Example:

The following example checks that user is not trying to schedule an appointment in the past:

```
::class 'OnlineAppointmentDlg' subclass ResDialog

::method ok

  dtp = self~newDateTimePicker(IDC_DTP_APPOINTMENT)

  if dtp~getDateTime < .DateTime~new then do
    msg = "Appointments can not be scheduled in the past!"
    ret = MessageDialog(msg, self~dlgHandle, "Appointment Error", "OK", "ERROR")
    return 0
  end

  return self~ok:super
```

3.8.5. newEdit

```
>>--newEdit(--id--)------------------------------><
```

The *newEdit* method returns an object of the *Edit* control class. This object represents the edit control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]

The resource ID of the edit dialog control. May be numeric or *symbolic*.

Return value:

An object of the edit control class or *.nil* on any error.

Example:

The following example validates that the user has filled in the last name field in a dialog form:

```
::class 'MyDlgClass' subclass RcDialog

::method validate
  editCtrl = self~newEdit(IDC_EDIT_LNAME)
  if editCtrl == .nil then return .false
  if editCtrl~getText~space(0) == "" then return .false
  else return .true
```

3.8.6. newGroupBox

```
>>--newGroupBox(--id--)--------------------------><
```

The *newGroupBox* method returns an object of the *GroupBox* control class. This object represents the group box control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]

The resource ID of the group box dialog control. May be numeric or *symbolic*.

Remarks:

Although a group box control is actually a type of a button control, usually they do not need to be manipulated after the underlying dialog is created and are often given a resource ID of -1. In order to obtain a new group box object, the underlying group box control in the dialog has to have a positive resource ID.

Return value:

An object of the group box control class or *.nil* on any error.

Example:

The following example shows how the text (label) of a group box could be changed during program execution:

```
::class 'MyPhoneBook' subclass ResDialog

::method onUseFull
  gb = self~newGroupBox(IDC_GB_TELEPHONE)
  chkBox = self~newCheckBox(IDC_CHK_USE_FULL)
  if chkBox~checked then do
    gb~setText("Phone Numbers (including area code)")
  end
  else do
    gb~setText("Phone Numbers")
  end
```

3.8.7. newListBox

```
>>--newListBox(--id--)---------------------------><
```

The *newListBox* method returns an object of the *ListBox* control class. This object represents the list box control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]

The resource ID of the list box dialog control. May be numeric or *symbolic*.

Return value:

An object of the list box control class or *.nil* on any error.

Example:

The following example updates the list box with a symbolic id of **`IDC_LB_AREAS`** by first deleting all items in the list box, then adding some new items and preselecting the item with the text of "City":

```
::class 'MyDlgClass' subclass RCDialog

::method updateList
  lb = self~newListBox(IDC_LB_AREAS)
  if lb == .nil then return 0
  lb~deleteAll
  lb~add("Town")
  lb~add("City")
  lb~add("Green")
  lb~add("Forest")
  lb~select("City")
```

3.8.8. newListView

```
>>--newListView(--id--)--------------------------><
```

The *newListView* method returns an object of the *ListView* control class. This object represents the list-view control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]

The resource ID of the list-view dialog control. May be numeric or *symbolic*.

Return value:

An object of the list-view control class or *.nil* on any error.

Example:

The following example :

```
::class 'MyDlgClass' subclass RcDialog

::method initDialog
  lc = self~newListView(IDC_LV_EMPLOYEES)
  if lc == .nil then return
  lc~~add(101222)~~add(,"Smith")~~add( ,"John")
  lc~~add(101223)~~add(,"Michael")~~add( ,"Carl")
```

3.8.9. newMonthCalendar

```
>>--newMonthCalendar(--id--)---------------------><
```

The *newMonthCalendar* method returns an object of the *MonthCalendar* control class. This object represents the month calendar control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]
: The resource ID of the month calendar dialog control. May be numeric or *symbolic*.

Return value:
An object of the month calendar control class or *.nil* on any error.

Example:
The following example is from a program that takes applications from people to participate in a clinical trial of a new drug. When the applicant agrees to the terms for participating, the program checks that the applicant meets the necessary qualifications. One of which is that the applicant be at least 50 years of age:

```
::method onAgree

  earliestAcceptableYear = .DateTime~new~addYears(-50)

  calendar = self~newMonthCalendar(IDC_MC_BIRTHDATE)

  if calendar~date > earliestAcceptableYear then do
    msg = "You must be 50 or over to participate in this clinical trial."
    ret = MessageDialog(msg, self~dlgHandle, "Not Qualified", "OK", "WARNING")
    return .false
  end

  ...  -- More checks

  return .true
```

3.8.10. newProgressBar

```
>>--newProgressBar(--id--)-----------------------><
```

The *newProgressBar* method returns an object of the *ProgressBar* control class. This object represents the progress bar control in the underlying dialog with the specified resource ID.

Arguments:
The single argument is:
id [required]
: The resource ID of the progress bar dialog control. May be numeric or *symbolic*.

Return value:
An object of the progress bar control class or *.nil* on any error.

Example:
The following example initializes the progress bar with symbolic resource ID to use a step value of 50 and sets its range to 1 through 2,147,483,647. As the program does its work, it updates the progress bar position incrementally based on the amount of work it calculates was done since the last update:

```
::class 'MyDlgClass' subclass ResDialog

::method initDialog
  pb = self~newProgressBar(IDC_PBAR_PROGRESS)
  if pb == .nil then return
  pb~setStep(50)
  pb~setFullRange(1, 2147483647)
```

```
::method updateProgress private
  use strict arg amount
  self~newProgressBar(IDC_PBAR_PROGRESS)~setPos(amount)
```

3.8.11. newPushButton

```
>>--newPushButton(--id--)------------------------><
```

The *newPushButton* method returns an object of the *Button* control class. This object represents the push button control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]

The resource ID of the push button dialog control. May be numeric or *symbolic*.

Return value:

An object of the button control class or *.nil* on any error.

Example:

The following example displays the state of the push button in the underlying dialog with the resource ID of 1:

```
::class 'MyDlgClass' subclass RcDialog

::method currentState
  pushButton = self~newPushButton(1)
  if pushButton == .nil then return .false
  say "State is" pushButton~state
  return .true
```

3.8.12. newRadioButton

```
>>--newRadioButton(--id--)-----------------------><
```

The *newRadioButton* method returns an object of the *RadioButton* control class. This object represents the radio button control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]

The resource ID of the radio button dialog control. May be numeric or *symbolic*.

Return value:

An object of the radio button control class or *.nil* on any error.

Example:

The following example displays a message if the radio button with a symbolic ID of IDC_RB_CHOICE is selected:

```
::class 'MyDlgClass' subclass ResDialog
```

```
::method currentState
  rb = self~newRadioButton(IDC_RB_CHOICE)
  if rb == .nil then return .false
  if rb~checked then say "The radio button is selected!"
  return .true
```

3.8.13. newReBar

```
>>--newReBar(--id--)------------------------><
```

The *newReBar* method returns an object of the *ReBar* control class. This object represents the rebar control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]

The resource ID of the rebar dialog control. May be numeric or *symbolic*.

Return value:

An object of the rebar control class or *.nil* on any error.

Example:

The following example gets a new rebar control object that represents the rebar control in the underlying dialog with the symbolic ID of IDC_RBAR. The application then begins populating the rebar with bands:

```

    rb = self~newReBar(IDC_RBAR)

    cb = self~newComboBox(IDC_CB)
    r = cb~windowRect

    band = .ReBarBandInfo~new(cb, 'Combo Box', CHILDEDGE, 'Bogus', 50)
    band~cxMinChild = 0
    band~cyMinChild = r~bottom - r~top

    ret = rb~insertBand(band)
```

3.8.14. newScrollBar

```
>>--newScrollBar(--id--)------------------------><
```

The *newScrollBar* method returns an object of the *ScrollBar* control class. This object represents the scroll bar control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]

The resource ID of the scroll bar dialog control. May be numeric or *symbolic*.

Return value:

An object of the scroll bar control class or *.nil* on any error.

Example:

The following example sets a new range and a new position for the horizontal scroll bar with the resource ID of 317:

```
::class 'MyDlgClass' subclass RcDialog

::method focusPage
  hScrollBar = self~newScrollBar(317)
  if hScrollBar == .nil then return .false
  hScrollBar~setRange(0, 1000, 0)
  hScrollBar~setPos(500, 1)
  return .true
```

3.8.15. newStatic

```
>>--newStatic(--id--)----------------------------><
```

The *newStatic* method returns an object of the *Static* control class. This object represents the static control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]

The resource ID of the static dialog control. May be numeric or *symbolic*.

Return value:

An object of the static control class or *.nil* on any error.

Remarks:

Often static controls are give a resource ID of -1 because they do not usually need to be changed after the underlying dialog is created. However, if the programmer needs to manipulate a static control in any way, then the control has to have a positive resource ID.

Example:

The following example gets a new static control object that represents the static control in the underlying dialog with the symbolic ID of IDC_ST_NAME. Provided the underlying dialog control exists, it is resized and its text, background color, and foreground color is changed:

```

  ::class 'MyDlgClass' subclass RcDialog

  ::method reArrange
    di = self~newStatic(IDC_ST_NAME)
    if di == .nil then return
    di~setRect(.Rect~new(0, 0, 100, 25), "NOMOVE HIDE")
    di~setText("Processing layout update!")
    di~setColor(7,4)
    di~show
    ...
```

3.8.16. newStatusBar

```
>>--newStatusBar(--id--)-------------------------><
```

The *newStatusBar* method returns an object of the *StatusBar* control class. This object represents the status bar control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]

The resource ID of the status bar dialog control. May be numeric or *symbolic*.

Return value:

An object of the status bar control class or *.nil* on any error.

Example:

The following example gets a new status bar control object that represents the status bar control in the underlying dialog with the symbolic ID of IDC_STATUSBAR. Provided the underlying dialog control exists, it is divided into parts and the text for each part is assigned and the text for each part is set:

```
    status = self~newStatusBar(IDC_STATUSBAR)

    parts = .array~of(50, 100, 200, -1)
    status~setParts(parts)
    status~setText("Line 1", 1)
    status~setText("Column 1", 2)
    status~setText("No Selection", 3)
    status~setText("RW", 4)
```

3.8.17. newTab

```
>>--newTab(--id--)-------------------------------><
```

The *newTab* method returns an object of the *Tab* control class. This object represents the tab control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]

The resource ID of the tab dialog control. May be numeric or *symbolic*.

Return value:

An object of the tab control class or *.nil* on any error.

Example:

The following example initializes the tab control with symbolic ID IDC_TAB to have 5 tabs:

```
::class 'MyDlgClass' subclass ResDialog

::method initDialog
  self~newTab(IDC_TAB)~addSequence("Design", "Implementation", -
                                   "Test", "Review", "Release")
```

3.8.18. newToolBar

```
>>--newToolBar(--id--)---------------------------><
```

The *newToolBar* method returns an object of the *ToolBar* control class. This object represents the toolbar control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]
: The resource ID of the toolbar dialog control. May be numeric or *symbolic*.

Return value:

An object of the toolbar control class or *.nil* on any error.

Details

Raises syntax errors when incorrect usage is detected.

Example:

The following example gets a *ToolBar* object that represents the underlying Windows toolbar with the symbolic ID of IDC_TOOLBAR and then begins initializing it:

```
::method initDialog
  expose tb

  tb = self~newToolBar(IDC_TOOLBAR)

  ret = tb~setExtendedStyle('DOUBLEBUFFER MIXEDBUTTONS')
  ret = tb~setBitmapSize(.Size~new(24, 24))

  self~loadBitmaps(tb)

  strings = .array~of("Press Me Please", "Where Am I?","Please Push Me", "Get Button",
 "Customize")
  ret = tb~addString(strings)

  tbb1 = .TbButton~new(IDB_PRESS, "Press Me Please", "BUTTON", "ENABLED", , 1); say 'tbb1
 text:' tbb1~text
  tbb2 = .TbButton~new(IDB_WHERE, 2, "BUTTON", "ENABLED", , 11)
  tbb3 = .TbButton~new(IDB_PUSH, 3, "BUTTON", "ENABLED", , 3)
  tbb4 = .TbButton~new(IDB_COPY_TO_FOLDER, 4, "BUTTON", "ENABLED", , 4)
  tbb5 = .TbButton~new(IDB_CUT_CLIPBOARD, 5, "BUTTON", "ENABLED", , 5)

  buttons = .array~of(tbb1, tbb2, tbb3, tbb4, tbb5)
  ret = tb~addButtons(buttons) -- Returns true on success, othewise false

  -- Now that the tool bar has its buttons, tell it to recalculate its size.
  tb~autoSize

  -- And show the tool bar.
  tb~show
```

3.8.19. newToolTip

```
>>--newToolTip(--id--)---------------------------><
```

The *newToolTip* method returns an object of the *ToolTip* class. This object represents the ToolTip control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]

The resource ID of the ToolTip dialog control. May be numeric or *symbolic*.

Return value:

Returns the Rexx ToolTip object that represents the Windows ToolTip control with the specified resource ID, or the **`.nil`** object on any error

Remarks:

The *newToolTip* method can not be invoked until the underlying ToolTip control has been created using the *createToolTip* method.

There is only one Rexx object created for each single Windows dialog control. Once the *newToolBar* method has been invoked once, each successive invocation of the *newToolBar* method, using the same resource ID, will return the same Rexx object.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example instantiates a new toolbar object and then begins to initialize the underlying Windows toolbar:

```
::method initDialog
  ...

  tb = self~newToolBar(IDC_TOOLBAR)

  ret = tb~setExtendedStyle('DOUBLEBUFFER MIXEDBUTTONS')
  ret = tb~setBitmapSize(.Size~new(24, 24))
  ...
```

3.8.20. newTrackBar

```
>>--newTrackBar(--id--)--------------------------><
```

The *newTrackBar* method returns an object of the *TrackBar* control class. This object represents the track bar control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]

The resource ID of the track bar dialog control. May be numeric or *symbolic*.

Return value:

An object of the track bar control class or *.nil* on any error.

Remarks:

Prior to the effort to *unify* the class and method names in ooDialog, the track bar control was called a slider. This may have been the most egregious misnomer in ooDialog and the term *slider* is no longer used.

Example:

In the following example, the initialization of the dialog controls is broken down into sub-steps to make the program more readable and easier to understand. The initialization of the a track bar control is shown:

```
::class 'MyDlgClass' subclass ResDialog

::method initDialog
  self~initTheListView
  self~initTheButtons
  self~initTheTrackBar

::method initTheTrackBar private
  trackBar = self~newTrackBar(IDC_TB_REPEAT_RATE)
  if trackBar == .nil then return

  trackBar~clearSelRange(.false)
  trackBar~setMax(200, .false)
  trackBar~setTickFrequency(50)
  trackBar~setTickAt(75)
  trackBar~setSelStart(20, .false)
  trackBar~setSelEnd(180, .true)
  trackBar~pos = 167
```

3.8.21. newTreeView

```
>>--newTreeView(--id--)--------------------------><
```

The *newTreeView* method returns an object of the *TreeView* control class. This object represents the tree view control in the underlying dialog with the specified resource ID.

Arguments:

The single argument is:

id [required]

The resource ID of the tree view dialog control. May be numeric or *symbolic*.

Return value:

An object of the tree view control class or *.nil* on any error.

Example:

The following example :

```
::class 'MyDlgClass' subclass ResDialog

::method initDialog

  tc = self~newTreeView(101)
  if tc == .nil then return

  tc~add("Root 1")
  tc~add(  ,"Item 1")
  tc~add(  ,"Item 2")
  tc~add(  ,"Item 3")
  tc~add("Root 2", , ,"EXPANDED")
  tc~add(  ,"Item 4", , ,"BOLD")
  tc~add(  ,"Item 5")
  tc~add(  ,"Subroot")
  tc~add(  ,    ,"Item 6",3)
```

3.8.22. newUpDown

```
>>--newUpDown(--id--)----------------------------><
```

The *newUpDown* method returns an object of the *UpDown* control class. This object represents the up down control in the underlying dialog with the specified resource ID.

Arguments:

The arguments are:

id [required]

The resource ID of the up down dialog control. May be numeric or *symbolic*.

Return value:

An object of the **UpDown** class or *.nil* on any error.

Example:

The following example initializes the up down control with symbolic ID IDC_UPD :

```
::method initDialog

  upDown = self~newUpDown(IDC_UPD)
  upDown~setRange(1, 20000)
  upDown~setPosition(64)
```

3.9. Connecting Event Methods

The dialog object methods that create a connection between a Windows *event* notification and a method of the Rexx dialog object are all implemented in the *EventNotification* class. These methods are documented here.

3.9.1. addUserMsg

```
EventNotification::addUserMsg


>>--addUserMsg(-methodName-,-winMsg-+------+-+----------+-+------+-+----------+-+------+-)-><
                                    +-,-f1-+ +-,-wParam-+ +-,-f2-+ +-,-lParam-+ +-,-f3-+
```

3.9.2. connectActivate

```
EventNotification::connectActivate


>>--connectActivate(--+--------------+--)--------><
                      +--methodName--+
```

3.9.3. connectAllSBEvents

```
EventNotification::connectAllSBEvents
```

```
>>--connectAllSBEvents(--id--,--methodName--+-------+-+-------+-+-------+--)---><
                                            +-,-min-+ +-,-max-+ +-,-pos-+
```

3.9.4. connectButtonEvent

```
EventNotification::connectButtonEvent


>>--connectButtonEvent(--id--,--event--+--------------+--)---------------------><
                                       +-,--methodName=+
```

3.9.5. connectComboBoxEvent

```
EventNotification::connectComboBoxEvent


>>--connectComboBoxEvent(--id--,--event--+--------------+--)-------------------><
                                         +-,--methodName=+
```

3.9.6. connectCommandEvents

```
EventNotification::connectCommandEvents


>>--connectCommandEvents(--id--,--methodName--)--><
```

3.9.7. connectDateTimePickerEvent

```
EventNotification::connectDateTimePickerEvent


>>--connectDateTimePickerEvent(--id--,--event--+---------+--+-------------+-)--><
                                               +-,-mName-+  +-,-willReply-+
```

3.9.8. connectDraw

```
EventNotification::connectDraw


>>--connectDraw--(--+-----+--+---------------+--)------------------------------><
                    +--id-+  +-,--methodName-+
```

3.9.9. connectEachSBEvent

```
EventNotification::connectEachSBEvent


>>--connectEachSBEvent(-id-,-mthWhenUp-,-mthWhenDown-+---------------+--------->
                                                     +-,-mthWhenDrag-+
>--+-------+-+-------+-+-------+--+----------+-+----------+-+----------+----->
   +-,-min-+-+-,-max-+ +-,-pos-+  +-,-mthPgUp-+ +-,-mthPgDn-+ +-,-mthTop-+

>--+-------------+-+------------+-+------------+-+-------------+--)------------><
   +-,-mthButtom-+ +-,-mthTrack-+ +-,-mthEndSc-+ +-,-willReply-+
```

3.9.10. connectEditEvent

```
EventNotification::connectEditEvent


>>--connectEditEvent(--id--,--event--+---------------+--)----------------------><
                                     +-,--methodName=+
```

3.9.11. connectFKeyPress

```
EventNotification::connectFKeyPress


>>--connectFKeyPress(--methodName--)-------------><
```

3.9.12. connectHelp

```
EventNotification::connectHelp


>>--connectHelp(--methodname--)------------------><
```

3.9.13. connectKeyPress

```
EventNotification::connectKeyPress


>>--connectKeyPress(--methodName--,--keys-+------------+--)--------------------><
                                          +-,--filter--+
```

3.9.14. connectListBoxEvent

```
EventNotification::connectListBoxEvent


>>--connectListBoxEvent(--id--,--event--+--------------+--)-------------------><
                                        +-,--methodName=+
```

3.9.15. connectListViewEvent

```
EventNotification::connectListViewEvent


>>--connectListviewEvent(--id--,--event--+--------------+--)------------------><
                                         +-,--methodName=+
```

3.9.16. connectMonthCalendarEvent

```
EventNotification::connectMonthCalendarEvent


>>--connectMonthCalendarEvent(--id--,--event--+---------+--+-------------+-)---><
                                              +-,-mName-+  +-,-willReply-+
```

3.9.17. connectMove

```
EventNotification::connectMove


>>--connectMove(--methodName--)------------------><
```

3.9.18. connectNotifyEvent

```
EventNotification::connectNotifyEvent


>>--connectNotifyEvent(--id--,--event--+--------------+--)--------------------><
                                       +-,--methodName=+
```

3.9.19. connectPosChanged

```
EventNotification::connectPosChanged


>>--connectPosChanged(--methodName--)------------><
```

3.9.20. connectResize

```
EventNotification::connectResize


>>--connectResize(--methodName--)----------------><
```

3.9.21. connectResizing

```
EventNotification::connectResizing


>>--connectResizing(--methodName--)--------------><
```

3.9.22. connectScrollBarEvent

```
EventNotification::connectScrollBarEvent
```

```
>>--connectScrollBarEvent(--id--,--event--+---------------+--)-----------------><
                                          +-,--methodName=+
```

3.9.23. connectSizeMoveEnded

```
EventNotification::connectSizeMoveEnded


>>--connectSizeMoveEnded(--methodName--+--------------+--)---------------------><
                                       +-,-willReply--+
```

3.9.24. connectStaticEvent

```
EventNotification::connectStaticEvent


>>--connectStaticNotify(--id--,--event--,-+---------------+--)-----------------><
                                          +-,--methodName-+
```

3.9.25. connectTabEvent

```
EventNotification::connectTabEvent


>>--connectTabEvent(--id--,--event--+---------------+--+--------------+--)-----><
                                    +-,--methodName-+  +-,-willReply--+
```

3.9.26. connectToolBarEvent

```
EventNotification::connectToolBarEvent


>>--connectToolBarEvent(--id--,--event--+-----------+--+--------------+--)-----><
                                        +-,-mthName-+  +-,-willReply--+
```

3.9.27. connectToolTipEvent

```
EventNotification::connectToolTipEvent


>>--connectToolTipEvent(--id--,--event--+-----------+--+--------------+--)-----><
                                        +-,-mthName-+  +-,-willReply--+
```

3.9.28. connectTrackBarEvent

```
EventNotification::connectTrackBarEvent


>>--connectTrackBarEvent(--id--,--event--+---------------+--)------------------><
                                         +-,--methodName-+
```

3.9.29. connectTreeViewEvent

```
EventNotification::connectTreeViewEvent


>>-connectTreeViewEvent(--id--,--event--+-------------+--+-------------+--)----><
                                        +-,--mthName--+  +-,-willReply-+
```

3.9.30. connectUpDownEvent

```
EventNotification::connectUpDownEvent


>>--connectUpDownEvent(--id--,--event--+--------------+--+-------------+-)-----><
                                       +-,-methodName-+  +-,-willReply-+
```

3.9.31. defListDragHandler

```
EventNotification::defListDragHandler


>>--defListDragHandler(--id--,--item--,--point--)-----------------------------><
```

3.9.32. defTreeDragHandler

```
EventNotification::defTreeDragHandler


>>--defTreeDragHandler(--id--,--item--,--point--)-----------------------------><
```

3.9.33. disconnectKeyPress

```
EventNotification::disconnectKeyPress


>>--disconnectKeyPress(--+--------------+--)-----><
                         +--methodName--+
```

3.9.34. hasKeyPressConnection

```
EventNotification::hasKeyPressConnection


>>--hasKeyPressConnection(--+--------------+--)--><
                            +--methodName--+
```

3.10. Standard Event Handling Methods

The ooDialog framework provides some default command *event* handling methods. At this time there are 3 handlers, one for the *ok*, the *cancel*, and the *help* command events. These 3 specific command events are generated when a *Button*, with the correct *resource ID*, is clicked or pushed. They can also be generated when a *menu* item with the correct resource ID is selected. The cancel event is also generated, in a dialog, when the user closes the dialog using the close icon in the title bar, or presses the escape key.

When a new dialog object is instantiated, ooDialog uses the *EventNotification* class to automatically connect the notifications for these events to the methods documented in this section. *Symbolic* IDs are also defined and placed in the *constDir*. This table summarizes these 3 events, their resource IDs, symbolic IDs, and event handling methods:

Table 3.3. Standard Event Handling

Event	Resource ID	Symbolic ID	Default Method
ok	1	IDOK	*ok*
cancel	2	IDCANCEL	*cancel*
help	9	IDHELP	*help*

Note that the help **command** event notification discussed here is not the same as the Windows help event notification connected through the *connectHelp*() method.

3.10.1. cancel

```
>>--cancel---------------------------------------><
```

A default *cancel* method is provided by the ooDialog framework. The **`cancel`** event notification is *automatically* connected to this method. This method is meant to be over-ridden by the programmer if the default implementation is not sufficient for the needs of the program.

The method is invoked in response to the **`cancel`** command event. The default implementation does the proper clean up to close the dialog and end its execution.

The programmer might want to over-ride the *cancel* method in her subclass. Perhaps to warn the user that there are unsaved changes for example.

If the programmer is going to end or close the dialog programmatically, it is *important* to always do so by invoking either the base class *ok* method or the base class *cancel* method. This ensures that the proper clean up is done. Failure to do so may produce unpredictable results.

If the *cancel* method is over-ridden in a subclass, invoke the super class's *cancel* method to close the dialog. Or return 0 to prevent the dialog from closing. The dialog continues executing if you return 0 without invoking of the super class's *cancel* method.

The *example* given for the *ok* method shows the proper way to over-ride the *ok* method. The general principle could be used to over-ride the *cancel* method as well.

3.10.2. help

```
>>--help-----------------------------------------><
```

A default *help* method is provided by the ooDialog framework. The **`help`** event notification is *automatically* connected to this method. This method is meant to be over-ridden by the programmer. The default implementation does nothing and returns 0. The programmer could over-ride this method to display some form of help to the user.

3.10.3. ok

```
>>--ok-------------------------------------------><
```

A default *ok* method is provided by the ooDialog framework. The **ok** event notification is *automatically* connected to this method. This method is meant to be over-ridden by the programmer if the default implementation is not sufficient for the needs of the program.

The method is invoked in response to the **ok** command event. The default implementation calls the *validate*() method to determine if the dialog should close or not. If *validate* returns true the dialog is closed. If it returns false, the dialog continues to execute. The default implementation of *validate* always returns true.

The programmer might want to over-ride the *ok* method in his subclass. In general, there is probably no need to. Anything done in the over-ridden *ok* method, could just as easily be done by overriding the *validate*() and / or the *leaving*() methods.

If the programmer is going to end or close the dialog programmatically, it is *important* to always do so by invoking either the base class *ok* method or the base class *cancel* method. This ensures that the proper clean up is done. Failure to do so may produce unpredictable results.

If the *ok* method is over-ridden in a subclass, invoke the super class's *ok* or *cancel* method (depending on circumstance) to close the dialog. Or return 0 to prevent the dialog from closing. The dialog continues executing if you return the value 0 with invoking one of the super class methods

Example:

The following example shows how to over-ride the *ok* method. It checks that the user really wants to quite.

```
::method ok unguarded

  dateTime = .DateTime~new
  currentTime = dateTime~civilTime

  buttons   = "OKCANCEL"
  defButton = "DEFBUTTON1"
  icon      = "INFORMATION"
  msg       = "It is now:" currentTime
  title     = "Quitting Now"

  if dateTime~hours < 16 then do
    break = .endOfLine~copies(2)

    icon = "WARNING"
    defButton = "DEFBUTTON2"
    chide = "It is NOT 4:00 pm." break "Are you SURE you should quit!"

    msg ||= break || chide
  end

  ret = messageDialog(msg, self~hwnd, title, buttons, icon, defButton)

  if ret == 1 then return self~ok:super
  else return 0
```

As noted above, rather than over-ride the *ok* method the same code could have been put into the *validate* method.

```
::method validate unguarded
```

```
dateTime = .DateTime~new
currentTime = dateTime~civilTime

buttons   = "OKCANCEL"
defButton = "DEFBUTTON1"
icon      = "INFORMATION"
msg       = "It is now:" currentTime
title     = "Quitting Now"

if dateTime~hours < 16 then do
  break = .endOfLine~copies(2)

  icon = "WARNING"
  defButton = "DEFBUTTON2"
  chide = "It is NOT 4:00 pm." break "Are you SURE you should quite!"

  msg ||= break || chide
end

ret = messageDialog(msg, self~hwnd, title, buttons, icon, defButton)

if ret == 1 then return .true
else return .false
```

3.10.4. leaving

```
>>--leaving--------------------------------------><
```

The *leaving* method is invoked automatically when the underlying Windows dialog is being closed. It is intended to be over-ridden by the programmer, if desired. The default implementation does nothing. Over-ridding the method gives the programmer a chance to do some final clean up.

3.10.5. validate

```
>>--validate-------------------------------------><
```

The *validate* method is a method that is intended to be over-ridden by the programmer. Its purpose would be to validate the user's input and decide whether or not to close the dialog. The default *ok* method implementation calls *validate*. If the return is true, the *ok* method will close the dialog. If the return is false, the dialog is not closed.

The default implementation of *validate* provided by the ooDialog framework does nothing and always returns true.

Example:

In the following example of an over-ride of *validate*, a check is done to determine if the edit control with resource ID 203 is empty. If it is empty, validate returns false, which indicates that the dialog cannot be closed.

```
::class 'MyDialog' subclass ResDialog

::method validate unguarded
   if self~newEdit(203)~getText = "" then return .false
   else return .true
```

3.11. Appearance and Behavior Methods

The methods listed in this section are related to the appearance or the behavior of the dialog or its controls. The section contains methods related to size, position, visibility, and title.

3.11.1. backgroundColor

```
>>--backgroundColor(--color--+---------+--)------><
                             +--isClr--+
```

Sets the background color of the dialog's *client area*. This is the color used to paint all areas of the dialog that are not covered by dialog controls.

Arguments:

The arguments are:

color [required]

Specifies the color to be used. This value can be either a palette color *index*, or a *COLORREF* number. To use a COLORREF number, the *isClr* argument must be true.

isClr [optional]

Specifies if the *color* argument is a COLORREF or a palette index. If true, the *color* argument is interpreted as a COLORREF, if false it is interpreted to be a palette index. The default if this argument is omitted is false.

Return value:

Returns true on success, false on error.

Remarks:

To correctly construct a COLORREF, it is easist to use the *colorRef* method of the *Image* class.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example sets the background color to a palish blue:

```
  self~backgroundColor(.Image~colorRef(171, 214, 245), .true)
```

3.11.2. backgroundSysColor

```
>>--backgroundSysColor(--color--)----------------><
```

Sets the background color of the dialog's *client area* to a system color. This is the color used to paint all areas of the dialog that are not covered by dialog controls.

Arguments:

The single argument is:

color [required]
: Specifies the system color to be used. This can be either the non-negative whole number ID or the keyword ID. IDs can be looked up in the System Color Elements *table*.

Return value:

Returns true on success, false on error.

Remarks:

System colors are the colors used for display elements in the system. These colors can be customized by the user. By using a system color ID, rather than using the *backgroundColor* method, the Rexx programmer can be sure the background matches that of the current system. Even if the user has changed from the normal default.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example sets the background color of the dialog to be the same as the color of a multiple document interface (MDI) application,

```
self~backgroundSysColor('APPWORKSPACE')
```

3.11.3. center

```
>>--center(--+------------+--)-------------------><
             +--showOpts--+
```

The center method moves the dialog to the screen center.

Arguments:

The only argument can be one of:

HIDEWINDOW
: Hides the dialog

SHOWWINDOW
: Shows the dialog

NOREDRAW
: Center the dialog without updating the display. Use the *update* method to manually update the display.

3.11.4. disableControl

```
>>--disableControl(--id--)-----------------------><
```

Disables the control with the specified resource id.

Arguments:

The only argument is:

id
: The *resource ID* of the control to be disabled. May be numeric or *symbolic*.

Return value:

The possible return values are:

true
: The method succeeded and the control was previously disabled.

false
: The method succeeded and the control was previously enabled.

-1
: The method failed, the *.systemErrorCode* is set with an error code.

Remarks:

When a control is enabled, it can gain the focus and will receive all keyboard or mouse input when it has the focus. The user can interact with the control.

When a control is disabled it receives no keyboard or mouse input. The user can not interact with it. The operating system draws the control with a visual style that indicates to the user that the control is disabled.

Details:

This method can not be used before the *underlying* dialog is created.

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

3.11.5. enableControl

```
>>--enableControl(--id--)------------------------><
```

Enables the control with the specified resource id.

Arguments:

The only argument is:

id
: The *resource ID* of the control to be enabled. May be numeric or *symbolic*.

Return value:

The possible return values are:

true
: The method succeeded and the control was previously disabled.

false
: The method succeeded and the control was previously enabled.

-1
: The method failed, the *.systemErrorCode* is set with an error code.

Remarks:

When a control is enabled, it can gain the focus and will receive all keyboard or mouse input when it has the focus. The user can interact with the control.

When a control is disabled it receives no keyboard or mouse input. The user can not interact with it. The operating system draws the control with a visual style that indicates to the user that the control is disabled.

Details:

This method can not be used before the *underlying* dialog is created.

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

3.11.6. ensureVisible

```
>>--ensureVisible--------------------------------><
```

The **`ensureVisible`** method causes the dialog to reposition itself so that the entire dialog is on the visible screen. If the entire dialog is already on the visible screen then no action is taken.

This is useful in a number of situations. It allows the programmer to move or resize the dialog and not have to worry that the dialog is off the screen. After finishing the move or resizing, the programmer can invoke the ensureVisible method and know that the dialog is entirely on the screen.

Arguments:

The method takes no arguments.

Return value:

This method always returns 0.

Example:

The following example reads in the previous size and position of the dialog and then opens the dialog in the position that the user had closed it. If the INI file is corrupted, or had been accidently edited, or a number of other things, it is possible that sizing and positioning the dialog using the values from the INI file will place the dialog of the screen. Invoking **`ensureVisible`** will cause the dialog to reposition itself so that it is completely on the screen, but only if needed. If the dialog is already completely on the screen, then no action is taken.

```
if \.useDefault then
    do
        -- Read oorexxtry.ini position & size the dialog based on its values
        handle = self~getSelf()
        k1 = SysIni('oorexxtry.ini','oorexxtry','k1')
        k2 = SysIni('oorexxtry.ini','oorexxtry','k2')
        k3 = SysIni('oorexxtry.ini','oorexxtry','k3')
        k4 = SysIni('oorexxtry.ini','oorexxtry','k4')
        if k1 = 'ERROR:' | k2 = 'ERROR:' | k3 = 'ERROR:' | k4 = 'ERROR:' then
            nop -- First execution will not find the ini file
        else
            do
                r = .Rect~new(k1, k2, k3-k1, k4-k2)
                self~setWindowRect(handle, r)
                self~ensureVisible
            end
    end
```

3.11.7. focusControl

```
>>--focusControl(--id--)-------------------------><
```

Sets the input focus to the specified dialog control.

Arguments:

The only argument is:

id [required]

The resource ID of the dialog control to set the focus to. May be numeric or *symbolic* .

Return value:

This method always returns 1.

3.11.8. getControlRect

```
DialogExtensions::getControlRect


>>--getControlRect(--id--)-----------------------><
```

3.11.9. getControlText

```
>>--getControlText(--id--)-----------------------><
```

The *getControlText* method gets the text of the specified dialog control.

Arguments:

The only argument is:

id [required]

The resource ID of the dialog control. May be numeric or *symbolic*.

Return value:

The text of the dialog control, which may be the empty string, or the empty string on error.

Remarks:

All windows can have text associated with them, although some times it is simply the empty string. For different types of windows the text serves different purposes and is called various things. For buttons the text is often called labels, for dialogs and main windows the text is often called the window title.

If the empty string is returned, the programmer can check the *.systemErrorCode* to determine if the text for the control is really the empty string, or if an error occurred.

Details:

Sets the *.systemErrorCode*.

3.11.10. getFocus

```
>>--getFocus-------------------------------------><
```

The *getFocus* method returns the window *handle* of the dialog control that currently has the input focus.

Arguments:

This method takes no arguments.

Return value:

The window handle of the dialog control that has the input focus, or 0 on error.

Details:

This method can not be invoked until the *underlying* Windows dialog has been created.

Raises syntax errors when incorrect usage is detected.

3.11.11. getWindowText

```
>>--getWindowText(--hwnd--)----------------------><
```

Gets the text of the specified window.

Arguments:

The arguments are:

hwnd [required]

The window *handle* whose text is to be gotten.

Return value:

On success returns the text of the window, which could be the empty string. On error returns the empty string.

Remarks:

The meaning of the *text* of a window varies with the type of window. For a dialog or other top-level window the text is the text of the title bar. For a button the text is the label, for an edit control the text is the contents of the control.

All windows can have text associated with them, although some times it is simply the empty string. If the empty string is returned, the programmer can check the *.systemErrorCode* to determine if the text for the window is really the empty string, or if an error occurred.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example is from a dialog that displays the full path name of the file being edited in the title bar of the dialog. The method checks the current tile of the dialog and if it does not match the file path name passed in to it, it resets the title.

```
::method updateCaption private
  use strict arg filePathName

  hwnd = self~hwnd
  titleBarText = self~getWindowText(hwnd)

  if titleBarText~caselessCompare(filePathName) <> 0 then do
```

```
    self~setWindowText(hwnd, filePathName)
  end
```

3.11.12. hideControl

```
>>--hideControl(--id--)--------------------------><
```

Removes the *visible* style from the specified dialog control and immediately redraws the area occupied by the control.

Arguments:

The only argument is:

id

The resource id of the control to be hidden, may be numeric or *symbolic*.

Return value:

The return values are:

1

The dialog control was previously visible.

0

The dialog control was already hidden, was already not visible.

-1

The resource ID is not valid.

Remarks:

When a window does not have the visible style, (when it is hidden,) and the area of the screen that the window occupies needs to be drawn, the operating system draws the window beneath the hidden window. The user can not interact with a hidden window, the window can not receive the keyboard or mouse input. However, the window still exists. All the other state of a hidden window remains unchanged and the programmer can still query or change that state. For instance, if an edit control is hidden, the programmer can still use the *getText* or *setText* methods to get or set the current text of the edit control.

The *hideControlFast* and *hideControl* methods are related. The only difference in the methods is that *hideControl* forces the operating system to immediately redraw the area occupied by the control while the *hideControlFast* does not redraw the area occupied by the control. The *hideControlFast* method is faster because no redrawing is done. Normally the *hideControlFast* method is used when a number of controls are hidden at one time. Then when all the controls are hidden, the programmer forces the area they occupied to be redrawn.

Details:

Sets the *.systemErrorCode*.

3.11.13. hideControlFast

```
>>--hideControlFast(--id--)----------------------><
```

Removes the *visible* style from the specified dialog control with no redrawing of the area occupied by the control.

Arguments:

The only argument is:

id

The resource id of the control to be hidden, may be numeric or *symbolic*.

Return value:

The return values are:

1

The dialog control was previously visible.

0

The dialog control was already hidden, was already not visible.

-1

The resource ID is not valid.

Remarks:

When a window does not have the visible style, (when it is hidden,) and the area of the screen that the window occupies needs to be drawn, the operating system draws the window beneath the hidden window. The user can not interact with a hidden window, the window can not receive the keyboard or mouse input. However, the window still exists. All the other state of a hidden window remains unchanged and the programmer can still query or change that state. For instance, if an edit control is hidden, the programmer can still use the *getText* or *setText* methods to get or set the current text of the edit control.

The *hideControl* and *hideControlFast* methods are related. The only difference in the methods is that *hideControl* forces the operating system to immediately redraw the area occupied by the control while the *hideControlFast* does not redraw the area occupied by the control. The *hideControlFast* method is faster because no redrawing is done. Normally the *hideControlFast* method is used when a number of controls are hidden at one time. Then when all the controls are hidden, the programmer forces the area they occupied to be redrawn.

Note that with the *hideControlFast* method, if the programmer does not manually force the area of the control to be redrawn at some point, the control will remain on the screen until some other event causes the operating system to redraw that portion of the screen. Also, the programmer needs to remember that it is the window beneath the control that needs to be redrawn. Having the hidden control redraw will not work. Since the control does not have the visible style, it does nothing.

Details:

Sets the *.systemErrorCode*.

3.11.14. hideWindow

```
>>--hideWindow(--hwnd--)-------------------------><
```

The hideWindow method hides a whole dialog window or a dialog item.

Arguments:

The only argument is:

hwnd

A handle to the window or dialog item. Use the *getSelf* or *getControlHandle* method to get a handle.

Example:

The following example gets the window handle of the top dialog (which is not necessarily the executing dialog, see the *get* method) and hides the whole dialog:

```
hwnd = MyDialog~get
MyDialog~hideWindow(hwnd)
```

3.11.15. hideWindowFast

```
>>--hideWindowFast(--hwnd--)---------------------><
```

The hideWindowFast method is similar to the *hideWindow* method, but it is faster because the window's or item's area is not redrawn. The hideWindowFast method is used when more than one state is modified. After the operations, you can manually redraw the dialog window, using the *update* method.

Arguments:

The only argument is:

hwnd

A handle to the window or dialog item

3.11.16. isMaximized

```
>>--isMaximized----------------------------------><
```

The **`isMaximized`** method is used to check if a dialog is currently maximized.

Arguments:

The method takes no arguments.

Return value:

.true

The dialog is maximized.

.false

The dialog is not maximized.

Example:

The following code snippet would check if the dialog is maximized and, if it is, restore it to the position and size it was prior to being maximized.

```
  if self~isMaximized then self~restore
```

3.11.17. isMinimized

```
>>--isMinimized----------------------------------><
```

The **`isMinimized`** method is used to check if a dialog is currently minimized.

Arguments:

The method takes no arguments.

Return value:

.true

The dialog is minimized.

.false

The dialog is not minimized.

Example:

The following example checks if the parent dialog is minimized and, if it is, the **`restore`** method is used to show the dialog in the size and position it was prior to being minimized. The complete program listing that this code snippet comes from is available (see the File Viewer *example*.)

```
::method cancel
  expose parent
  parent~enableControl(IDC_PB_VIEW)
  if parent~isMinimized then parent~restore
  return self~cancel:super
```

3.11.18. maximize

```
>>--maximize-------------------------------------><
```

The **`maximize`** method maximizes the dialog on the screen. This is the identical to the user clicking the maximize button on the dialog window. This method will maximize the dialog even if it does not have the **`MAXIMIZEBOX`** style when it is created.

This is a convenience method. It is functionally equivalent to using the *show* method with the **`MAX`** keyword.

Arguments:

The method takes no arguments.

Return value:

0

Maximizing was successful.

1

Maximizing failed.

Example:

The following code snippet minimizes the dialog to the taskbar. It is used in the FileViewer example. The complete program listing is available (see the File Viewer *example*.)

```
::method onView
  expose viewDlg editCntrl

  ...

  if viewDlg~initCode = 0 then do
    ...

    viewDlg~maximize
    self~minimize
  end
```

3.11.19. minimize

```
>>--minimize-------------------------------------><
```

The **`minimize`** method minimizes the dialog to the taskbar. This is the identical to the user clicking the minimize button on the dialog window. This method will minimize the dialog even if it does not have the **`MINIMIZEBOX`** style when it is created.

This is a convenience method. It is functionally equivalent to using the *show* method with the **`MIN`** keyword.

Arguments:

The method takes no arguments.

Return value:

0

Minimizing was successful.

1

Minimizing failed.

Example:

This example creates a secondary dialog to display the contents of a file. The secondary dialog is show full screen (maximized.) At the same time the parent dialog is minimized to the taskbar. The complete program listing is available (see the File Viewer *example*.)

```
::method onView
  expose viewDlg editCntrl

  fileName = editCntrl~getText
  viewDlg = .Viewer~new( , "fileView.h", self, fileName)
  if viewDlg~initCode = 0 then do
    self~disableControl(IDC_PB_VIEW)

    viewDlg~create(30, 30, 170, 180, "Viewer", "MAXIMIZEBOX MINIMIZEBOX")
    viewDlg~popUpAsChild(self, "HIDE", , IDI_DLG_APPICON)

    -- The underlying Windows dialog has to be created before it can be maximized.
    j = SysSleep(.1)

    viewDlg~maximize
    self~minimize
  end
```

3.11.20. moveControl

```
DialogExtensions::moveControl


>>--moveControl(--id--,--xPos--,--yPos--+-----------+--)----------------------><
                                        +-,-showOpt--+
```

3.11.21. resizeControl

```
DialogExtensions::resizeControl


>>--resizeControl(--id--,--width--,--height--+------------+--)----------------><
                                             +-,-showOpt--+
```

3.11.22. restore

```
>>--restore----------------------------------------><
```

The **`restore`** method restores a minimized or maximized dialog to its original position. This is the identical to the user taking action to restore the dialog window. The programmer can use this method to restore a dialog window that he previously minimized or maximized.

This is a convenience method. It is functionally equivalent to using the *show* method with the **`RESTORE`** keyword.

Arguments:

The method takes no arguments.

Return value:

0

Restoring was successful.

1

Restoring failed.

Example:

The following example checks if the parent dialog is minimized and, if it is, the **`restore`** method is used to show the dialog in the size and position it was prior to being minimized. The complete program listing for this code snippet is available (see the File Viewer *example*.)

```
::method cancel
  expose parent
  parent~enableControl(IDC_PB_VIEW)
  if parent~isMinimized then parent~restore
  return self~cancel:super
```

3.11.23. setControlColor

```
DialogExtensions::setControlColor


>>--setControlColor(--id--,--bk--+-------+--)----><
                                 +-,-fg--+
```

3.11.24. setControlFont

```
DialogExtensions::setControlFont
```

```
>>--setControlFont(--id--,--fonthandle--+-----------+--)-----------------------><
                                        +-.-redraw--+
```

3.11.25. setControlSysColor

```
DialogExtensions::setControlSysColor


>>--setControlSysColor(--id--,--bk--+-------+--)-><
                                    +-,-fg--+
```

3.11.26. setControlText

```
>>--setControlText(--id--,--text--)--------------><
```

The *setControlText* method sets the text of the specified dialog control to that specified.

Arguments:

The arguments are:

id [required]

The resource ID of the dialog control. May be numeric or symbolic.

text [required]

The new text for the dialog control.

Return value:

The return codes can be:

0

Success

-1

Error.

Remarks:

All windows can have text associated with them, although some times it is simply the empty string. For different types of windows the text serves different purposes and is called various things. For buttons the text is often called labels, for dialogs and main windows the text is often called the window title.

Details:

Sets the *.systemErrorCode*.

3.11.27. setFocus

```
>>--setFocus(--hwnd--)---------------------------><
```

The *setFocus* method sets the input focus to a dialog control specified by *hwnd* and returns the window handle of the control that previously had the focus.

Arguments:

The only argument is:

hwnd [required]

The window handle of the dialog control that is to receive the input focus.

Return value:

Returns the window handle of the dialog control that previously had the focus, or 0 if the focus was changed but the previous focus could not be determined. If an error is detected, -1 is returned.

Remarks:

If the *hwnd* argument is not the window handle of a dialog control in this dialog, the result is unpredictable.

This method can not be used before the *underlying* dialog is created.

Details:

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

3.11.28. setFocusToWindow

```
>>--setFocusToWindow(--hwnd--)-------------------><
```

The *setFocusToWindow* method moves the input focus to another top-level window or dialog. This has the effect of bringing that window to the foreground. This method returns a handle to the dialog *control* in this dialog that previously had the input focus.

Arguments:

The only argument is:

hwnd [required]

The window handle of the top-level window or dialog that is to receive the input focus.

Return value:

Returns the window handle of the dialog control that previously had the focus, or 0 if the focus was changed but the previous focus could not be determined. If an error is detected, -1 is returned.

Remarks:

If the *hwnd* argument is not the window handle of a top-level window or a dialog, the result is unpredictable.

This method can not be used before the *underlying* dialog is created.

Details:

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

3.11.29. setForegroundWindow

```
DialogExtensions::setForegroundWindow
```

```
>>--setForegroundWindow(--hwnd--)----------------><
```

3.11.30. setGroup

```
>>--setGroup(--id--+-----------+--)--------------><
                   +-wantStyle-+
```

Adds or removes the *group* style for the control specified.

Arguments:

The arguments are:

id [required]

The resource ID of the dialog control that will gain or lose the group style. May be numeric or *symbolic* .

wantStyle [optional]

A boolean (.true or .false) to indicate whether the control should have or not have the group style. True (the default) indicates the dialog control should have the group style and false indicates the control should not have the style.

Return value:

Negative values indicate the function failed, positive values indicate success.

Less than -1

The value is the negated operating system error code.

-1

The resource ID specified for the dialog control is not correct.

Greater than 0

The window style of the dialog control prior to adding or removing the group style.

Remarks:

The group style controls how the user can navigate through the dialog using the keyboard. For most dialogs this does not change while the dialog is executing. However, in some dialogs the programmer may want to change the navigation depending on the options the user selects.

The negative return values less than -1 are a hold over from a period prior to the introduction of the *.systemErrorCode*. This method now sets the **`.systemErrorCode`**, making it easier to use the **`.systemErrorCode`** value to look up the actual error.

Details:

Raises syntax errors when incorrect arguments, other than the resource ID, are detected.

Sets the *.systemErrorCode*.

3.11.31. setTabStop

```
>>--setTabStop(--id--,--+-----------+--)---------><
                        +-wantStyle-+
```

Add or remove the *tabstop* style for the specified control. When a control has the tabstop style, the user can set the focus to the control by using the tab key. When a control does not have this style, the tab key will skip over the control. Adding or removing this style during the execution of a dialog allows the programmer to alter how the user navigates through the dialog controls.

Arguments:

The arguments are:

id

The resource ID of the dialog control that will gain or lose the tabstop style.

wantStyle

A boolean (.true or .false) to indicate whether the dialog control should have or not have the tabstop style. True (the default) indicates the control should have the tabstop style and false indicates the control should not have the style.

Return value:

Negative values indicate the function failed, non-negative values indicate success.

-4 or less

The value is the negated Operating System Error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

-3

The second argument to the method is not a boolean.

-2

There is an (internal) problem with the dialog or the dialog handle.

-1

The resource ID of the control is not correct.

0 or greater

The window style of the dialog control prior to adding or removing the tabstop style.

3.11.32. setWindowRect

```
DialogExtensions::setWindowRect


Form 1:

>>--setWindowRect(--hwnd--,--rectangle--+---------+--)-------------------------><
                                        +-,-opts--+

Form 2:

>>--setWindowRect(--hwnd--,--pt--,--size--+---------+--)-----------------------><
                                          +-,-opts--+

Form 3:

>>--setWindowRect(--hwnd--,--x-,--y-,--cx-,--cy--+---------+--)----------------><
                                                 +-,-opts--+

Generic form:

>>--setWindowRect(--hwnd--,--rectCoordinates--+---------+--)-------------------><
                                              +-,-opts--+
```

3.11.33. setWindowText

```
>>--setWindowText(--hwnd--,--text--)-------------><
```

Sets the text for the specified window.

Arguments:

The arguments are:

hwnd [required]

The window *handle* whose text is to be set.

text [required]

The text to set for the window. May be the empty string.

Return value:

Returns 0 on success, 1 on error

Remarks:

The meaning of the *text* of a window varies with the type of window. For a dialog or other top-level window the text is the text of the title bar. For a button the text is the label, for an edit control the text is the contents of the control.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example is from a dialog that displays the full path name of the file being edited in the title bar of the dialog. The method checks the current tile of the dialog and if it does not match the file path name passed in to it, it resets the title.

```
::method updateCaption private
  use strict arg filePathName

  hwnd = self~hwnd
  titleBarText = self~getWindowText(hwnd)

  if titleBarText~caselessCompare(filePathName) <> 0 then do
    self~setWindowText(hwnd, filePathName)
  end
```

3.11.34. show

```
>>--show(--+--------+--)-------------------------><
           +--opts--+
```

Sets the dialog window's show state.

Argument:

The single, optional, *opts* argument can be zero or one of the following keywords, case is not significant:

NORMAL
: Makes the dialog visible in its default position and window size. This has the effect of restoring the dialog size and position if it is minimized or maximized. This is the default if the argument is omitted.

DEFAULT
: DEFAULT is an alias for NORMAL. The two keywords are functionally identical.

SHOWTOP
: Makes the dialog visible and the topmost window. Note that the dialog size and position are not restored if the dialog is minimized or maximized.

HIDE
: Makes the dialog invisible.

MIN
: Minimizes the dialog and activates the next window in the window order.

MAX
: Maximizes, and makes visible if necessary, the dialog.

INACTIVE
: Makes the dialog visible without changing the active window. When the NORMAL keyword is used, the dialog is shown and becomes the active window. The INACTIVE keyword makes the dialog visible without changing the focus from the current active window.

RESTORE
: Makes the dialog visible and restores it to its original size and position if it was minimized or maximized. An application should specify this flag when restoring a minimized window.

Remarks:

The *show* method is invoked automatically by the methods that run a dialog, *execute*, *popup*, etc.. Once the dialog is running this method can be used to hide, maximize, or otherwise change the show state as described by the keywords. There are also a number of convenience methods that do similar things, *hide*, *display*, *minimize*, *restore*, and so on.

The *show* method over-rides the *show* method of the *Window* class, which takes no arguments. The primary purpose of the dialog object *show* method is to allow flexibility in how the dialog is initially shown when it is run.

Return value:

Returns true if the window was previously visible and false if the window was previously hidden.

Example:

The following statement makes the dialog invisible:

```
dlg~show("HIDE")
```

3.11.35. showControl

```
>>--showControl(--id--)--------------------------><
```

The showControl method makes the given dialog item reappear on the screen.

Arguments:

The only argument is:

id

The ID of the item

3.11.36. showControlFast

```
>>--showItemFast(--id--)-------------------------><
```

The showControlFast method shows an item without redrawing its area. It is the counterpart to the *hideControlFast* method.

3.11.37. showWindow

```
>>--showWindow(--hwnd--)-------------------------><
```

The showWindow method shows the window or item again.

Arguments:

The only argument is:

hwnd

The handle of a window or an item

3.11.38. showWindowFast

```
>>-showWindowFast(--hwnd--)----------------------><
```

The showWindowFast method is the counterpart to the *hideWindowFast* method.

3.11.39. tabToNext

```
>>--tabToNext------------------------------------><
```

Sets the focus to the next tab stop dialog control in the dialog and returns the window *handle* of the dialog control that currently has the focus.

Arguments:

This method has no arguments.

Return value:

On success, the window handle of the dialog control that previously had the focus. Otherwise, 0, if the focus was changed, but the previous focus could not be determined. Returns -1 if an error is detected.

Remarks:

This method can not be used before the *underlying* dialog is created.

This method performs the same action as if the user pressed the tab key in the dialog.

Although this method sets the **`.systemErrorCode`** variable, there are no system errors that would change it from 0.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

3.11.40. tabToPrevious

```
>>--tabToPrevious--------------------------------><
```

Sets the focus to the previous tab stop dialog control in the dialog and returns the window *handle* of the dialog control that currently has the focus.

Arguments:

This method has no arguments.

Return value:

On success, the window handle of the dialog control that previously had the focus. Otherwise, 0, if the focus was changed, but the previous focus could not be determined. Returns -1 if an error is detected.

Remarks:

This method can not be used before the *underlying* dialog is created.

This method performs the same action as if the user pressed the shift-tab key in the dialog.

Although this method sets the **`.systemErrorCode`** variable, there are no system errors that would change it from 0.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

3.11.41. toTheTop

```
>>--toTheTop-------------------------------------><
```

The toTheTop method makes the dialog the topmost dialog.

Example:

The following example uses the *toTheTop* method to make the user aware of an alarm event. The **`Message`** and **`Alarm`** classes are built-in classes of Object Rexx. See the *Open Object Rexx: Reference* for further information.

```
aDialog = .MyDialog~new
msg = .Message~new(aDialog, "Remind")
a = .Alarm~new("17:30:00", msg)

::class MyDialog subclass ResDialog
      .
      .
      .
::method remind
  self~setControlText(102, "Don&apos;t forget to go home!")
  self~toTheTop
```

3.11.42. FileViewer Example Program

The FileViewer program is a complete working program that uses many of the methods of the base dialog. Portions of this program are presented as examples in the documentation for these methods. The complete program is shown here as a reference to how the code snippets all fit together.

The documentation for the individual methods used in the program has additional commentary that will help in understanding how the program works.

- *new*: This program uses a header file, **`fileView.h`** to define symbolic IDs for the dialog controls. The name of the file is one of the arguments used to create a new instance of a dialog object. (As in all Rexx classes, the arguments used in the *new* method are passed on to the *init* method of the class.)

- *execute*: Two dialogs are created in the example. The first dialog allows the user to enter the name of a file to view. Then the file itself is displayed in a second dialog. Each dialog uses a different application icon. The **`execute`** method documentation has more detail on the application icon.

- *initDialog* This method is called after the underlying Windows dialog has been created, but before it is shown on the screen. In the viewer dialog the **`init`** method is over-ridden and the text of the read-only multi-line edit control is set to the contents of the file. In both dialogs, this method is used to get and save a reference to the edit control of the dialog.

- *disableControl*: When the user clicks the "View File" button, the button is disabled until the user is finished viewing the file. When a button is disabled it can not be clicked by the user.

- *enableControl*: When the user closes the secondary viewer dialog, the "View File" button is enabled so the user can choose to view another file.

- *popupAsChild*: The viewer dialog is executed using this method so that both dialogs run independently, both dialogs stay enabled. The **`HIDE`** keyword is used for the show argument. This creates the dialog as invisible and prevents unnecessary screen flicker.

- *maximize*: Note that there is a 100 millisecond sleep right before the **`maximize method`** is invoked. This allows Windows to finish creating the dialog. The **`initDialog`** method will have already finished executing and the contents of the file are loaded into the multi-line edit control. Then, **`maximize`** resizes the dialog to take up the whole screen and shows it. The dialog appears on the user's screen, with the file displayed, in one screen drawing. This reduces the flicker on the screen.

- *minimize*: When the viewer dialog is maximized, the main dialog is minimized to the task bar.

- *initAutoDetection*: Because the contents of the file are loaded into the multi-line edit control during **`initDialog`**, auto detection must be turned off. Otherwise, after **`initDialog`** finishes executing, auto detection would set the empty string as the text for the control. The program overrides this method to turn auto detection off.

- *noAutoDetection*: Used in turning auto detection off.

- *connectResize*: When the viewer dialog is resized, the **`onSize`** method is invoked. The size of the multi-line edit control is then changed so it completely takes up the client area of the dialog. (The client area of a window is where all the child windows are drawn. In this case the edit control is the only child window.) Since the dialog is not resizable by the user (it does not have a sizing border) the only time the size can change is when the dialog is maximized, minimized, or restored.

- *hideControl*: The edit control in the viewer dialog is created invisible. Again, this helps reduce flicker.
- *getControlHandle*: The window handle of the edit control is obtained by this method ...
- *getClientRect*: ... then the size of the client area of the viewer dialog is obtained using this method. Then ..
- *setWindowRect*: ... the size and position of the edit control window is set to take up the entire dialog's client area.
- *showControl*: The **`showControl`** method is used to make the edit control visible when the dialog is initially shown.
- *isMinimized*: When the viewer dialog is closed, this method checks to see if the main dialog is still minimized. Since the dialogs run independently the user may have already restored this dialog.
- *restore*: If the main dialog is still minimized, then this method restores it to its normal position.

```
/* fileView.h  Simple symbolic ID definitions  */

#define IDD_DIALOG1          100
#define IDC_ST_TYPE          105
#define IDC_ENTRYLINE        106
#define IDC_MULTILINE        107
#define IDC_PB_VIEW          111

/* FileViewer.rex  Simple Dialog to view files full screen */

  dlg = .FileView~new( , "fileView.h")
  if dlg~initCode = 0 then do
    dlg~createCenter(170, 90, "The File Viewer Dialog", "VISIBLE MAXIMIZEBOX MINIMIZEBOX")
    dlg~execute("SHOWTOP", IDI_DLG_OOREXX)
  end

-- End of entry point.
::requires "ooDialog.cls"

::class FileView subclass UserDialog inherit AdvancedControls

::method defineDialog

  self~createStaticText(IDC_ST_TYPE, 10, 25, 150, 10, "", " Enter the name of a file to
 view:")
  self~createEdit(IDC_ENTRYLINE, 10, 35, 150, 10, "AUTOSCROLLH")

  -- When the view button is pushed, another dialog will show the file.
  self~createPushButton(IDC_PB_VIEW, 10, 55, 35, 15, "DEFAULT GROUP", "View File", onView)
  self~createPushButton(IDOK, 130, 55, 35, 15, , "Quit")

::method initDialog
  expose editCntrl
  editCntrl = self~newEdit(IDC_ENTRYLINE)

::method onView
  expose viewDlg editCntrl

  fileName = editCntrl~getText
  viewDlg = .Viewer~new( , "fileView.h", self, fileName)
  if viewDlg~initCode = 0 then do
    self~disableControl(IDC_PB_VIEW)
    viewDlg~create(30, 30, 170, 180, "Viewer", "MAXIMIZEBOX MINIMIZEBOX")
    viewDlg~popUpAsChild(self, "HIDE", , IDI_DLG_APPICON)

    -- The underlying Windows dialog has to be created before it can be maximized.
    j = msSleep(100)

    viewDlg~maximize
    self~minimize
  end

::class 'Viewer' subclass UserDialog inherit AdvancedControls

::method init
  expose parent filename
  use arg data, header, parent, fileName
  forward class (super)

::method initAutoDetection
  self~noAutoDetection

::method defineDialog
  expose wasMinimized

  wasMinimized = .false
  style = "VSCROLL HSCROLL MULTILINE READONLY"
  self~createEdit(IDC_MULTILINE, 0, 0, 170, 180, style, "cEntry")
  self~connectResize("onSize")

::method initDialog
  expose editControl fileName isHidden

  self~hideControl(IDC_MULTILINE)
  isHidden = .true

  editControl = self~newEdit(IDC_MULTILINE)
  fObj = .stream~new(fileName)
  text = fObj~charin(1, fObj~chars)
  fObj~close
  if text == "" then text = "   No file  "
  editControl~setText(text)

::method onSize
  expose wasMinimized
  use arg sizeEvent, sizeInfo

  if sizeEvent = 1 then wasMinimized = .true

  if sizeEvent = 0 |  sizeEvent = 2 then do
    if \ wasMinimized then self~resizeEditControl
    wasMinimized = .false
  end

::method resizeEditControl
  expose editControl isHidden

  hwnd = self~getControlHandle(IDC_MULTILINE)
  rect = self~clientRect

  self~setWindowRect(hWnd, rect)

  if isHidden then do
    self~showControl(IDC_MULTIINE)
    isHidden = .false
  end

::method cancel
```

```
  expose parent
  parent~enableControl(IDC_PB_VIEW)
  if parent~isMinimized then parent~restore
  return self~cancel:super
```

3.12. Bitmap Methods

The methods listed in this section deal with using bitmaps.

3.12.1. backgroundBitmap

```
>>--backgroundBitmap(--bmpFilename--+-------------+--)------------------------><
                                    +-,--"USEPAL"-+
```

The backgroundBitmap method sets a bitmap as the dialog's background picture.

Arguments:

The arguments are:

bmpFilename

The name of a bitmap file

option

Set the last argument to USEPAL if you want to use the color palette of the bitmap. See *installBitmapbutton* for more information.

3.12.2. changeBitmapButton

```
DialogExtensions::changeBitmapButton


>>--changeBitmapButton(--id--,--normal--+-----------+--+-----------+---------->
                                        +-,-focused-+  +-,-selected-+

>--+-----------+--+---------+--)----------------------------------------------><
   +-,-disabled-+  +-,-style-+
```

3.12.3. dimBitmap

```
DialogExtensions::dimBitmap


>>--dimBitmap(--id-,-bmp-,-cx-,-cy--+---------+--+---------+--+---------+--)---><
                                    +-,-stepX-+  +-,-stepY-+  +-,-steps-+
```

3.12.4. displaceBitmap

```
DialogExtensions::displaceBitmap


>>--displaceBitmap(--id--,--x--,--y--)----------><
```

3.12.5. drawBitmap

```
DialogExtensions::drawBitmap


>>--drawBitmap(--+-------+--,--id--+-----+--+-----+--+--------+--+--------+---->
                 +--hwnd-+         +-,-x-+ -+-,-y-+ -+-,-bmpX-+  +-,-bmpY-+

>--+---------+--+----------+--)------------------------------------------------><
   +-,-width-+  +-,-height-+
```

3.12.6. getBitmapPosition

```
DialogExtensions::getBitmapPosition


>>--getBitmapPosition(--id--,--pos--)------------><
```

3.12.7. getBitmapSize

```
DialogExtensions::getBitmapSize


>>--getBitmapSize(--id--)------------------------><
```

3.12.8. getBitmapSizeX

```
DialogExtensions::getBitmapSizeX


>>--getBitmapSizeX(--id--)-----------------------><
```

3.12.9. getBitmapSizeY

```
DialogExtensions::getBitmapSizeY


>>--getBitmapSizeX(--id--)-----------------------><
```

3.12.10. getBmpDisplacement

```
DialogExtensions::getBmpDisplacement


>>--getBmpDisplacement(--id--)-------------------><
```

3.12.11. installAnimatedButton

```
DialogExtensions::installAnimatedButton
```

```
>>--installAnimatedButton(--id,--+-------+-+--------+-,-bmpF-+--------+-,-mX-,-mY-->
                                 +-,-mth-+ +-,-cls--+        +-,-bmpT-+

>--+---------+--+---------+--,--delay---+--------+--+--------+--)--------------><
   +-,-sizeX-+  +-,-sizeY-+             -,+-xNow-+  +-,-yNow-+
```

3.12.12. installBitmapButton

```
DialogExtensions::installBitmapbutton


>>--installBitmapButton(-id-+-------+-,-bN--+------+-+------+-+-------+-+--------+-)-><
                            +-,-mth-+       +-,-bF-+ +-,-bS-+ +-,-bD--+ +-,-opts-+
```

3.12.13. scrollBitmapFromTo

```
DialogExtensions::scrollBitmapFromTo


>>--scrollBitmapFromTo(--id--,--fromX--,--fromY--,--toX--,--toY---------------->


>--+---------+--+---------+--+---------+--+------------+--)--------------------><
   +-,-stepX-+  +-,-stepY-+  +-,-delay-+  +-,-displace-+
```

3.12.14. scrollButton

```
DialogExtensions::scrollButton


>>--scrollButton(--id--,--xPos--,--yPos--,--lft--,--top--,--rght--,--bttm--)---><
```

3.12.15. setBitmapPosition

```
DialogExtensions::setBitmapPosition


Form 1:

>>--setBitmapPosition(--id--,--point--)----------><

Form 2:

>>--setBitmapPosition(--id--,--x--,--y--)--------><

Generic form:

>>--setBitmapPosition(--id--,--newPosition--)----><
```

3.12.16. tiledBackgroundBitmap

```
>>--tiledBackgroundBitmap(--bmpFilename--)-------><
```

The tiledBackgroundBitmap method sets a bitmap as the background brush (Windows NT® only). If the bitmap size is less than the size of the background, the bitmap is drawn repetitively.

Arguments:

The only argument is:

bmpFilename

The name of a bitmap file

3.13. Combo Box Methods

The following methods belong to combo boxes.

3.13.1. addComboEntry

```
>>--addComboEntry(--id--,--aString--)------------><
```

The addComboEntry method adds a string to the list of a combo box. The new item becomes the last one, if the list does not have the SORT flag set. In the case of a sorted list, the new item is inserted at the proper position.

Arguments:

The arguments are:

id

The ID of a combo box.

aString

The data to be inserted as a new line.

Example:

The following example adds the new line, **`Another item`**, to the list of combo box 103:

```
MyDialog~addComboEntry(103, "Another item")
```

3.13.2. changeComboEntry

```
>>--changeComboEntry(--id--,--+-------+--,--aString--)-------------------------><
                              +-index-+
```

The changeComboEntry method changes the value of a given entry in a combo box to a new string.

Arguments:

The arguments are:

id

The ID of the combo box

index

The index number of the item you want to replace. To retrieve the index, use the findComboEntry or getCurrentComboIndex method (see page *findComboEntry* or *getCurrentComboIndex*).

aString
: The new text.

Example:

In the following example, method **`ChangeButtonPressed`** changes the currently selected line of combo box 230 to the value in entry line 250:

```
     .
     .
     .
::method ChangeButtonPressed
   idx = self~GetCurrentComboEntry(230)
   str = self~getEditData(250)
   self~changeComboEntry(230, idx, str)
```

3.13.3. comboAddDirectory

```
>>--comboAddDirectory(--id--,--drvpath--,--fileAtrs--)---------><
```

The comboAddDirectory method adds all or selected file names in the given directory to the combo box.

Arguments:

The arguments are:

id
: The ID of the combo box.

drvpath
: The drive, path, and name pattern.

fileAtrs
: Specify the file attributes that the files must have in order to be added:

  READWRITE
  : Normal read/write files (same as none).

  READONLY
  : Files that have the read-only bit.

  HIDDEN
  : Files that have the hidden bit.

  SYSTEM
  : Files that have the system bit.

  DIRECTORY
  : Files that have the directory bit.

  ARCHIVE
  : Files that have the archive bit.

Example:

The following example fills the combo box list with the names of all read/write files with extension .REX in the given directory:

```
MyDialog~comboAddDirectory(203, drive":\"path"\*.rex", "READWRITE")
```

3.13.4. comboDrop

```
>>--comboDrop(--id--)----------------------------><
```

The comboDrop method deletes all items from the list of the given combo box.

Arguments:

The only argument is:

id

The ID of the combo box.

3.13.5. deleteComboEntry

```
>>--deleteComboEntry(--id--,--index--)-----------><
```

The deleteComboEntry method deletes a string from the combo box.

Arguments:

The arguments are:

id

The ID of the combo box.

index

The line number of the item to be deleted. Use the findComboEntry method (see *findComboEntry*) to retrieve the index of an item.

Example:

The following example shows a method that deletes the item that is passed to the method in the form of a text string from combo box 203:

```
      .
      .
      .
::method DeleteFromCombo
   use arg delStr
   idx = self~findComboEntry(203, delStr)
   self~deleteComboEntry(203, idx)
```

3.13.6. findComboEntry

```
>>--findComboEntry(--id--,--aString--)-----------><
```

The findComboEntry method returns the index corresponding to a given text string in the combo box.

Arguments:

The arguments are:

id
: The ID of the combo box

aString
: The string of which you search the index in the combo box.

Example:

See *deleteComboEntry* for an example.

3.13.7. getComboEntry

```
>>--getComboEntry(--id--,--index--)--------------><
```

The getComboEntry method returns the string at index of the combo box.

Arguments:

The arguments are:

id
: The ID of the combo box

index
: The index of the list entry to be retrieved

Example:

```
if dlg~getComboEntry(203,5)="JOHN"
then ...
```

3.13.8. getComboItems

```
>>--getComboItems(--id--)------------------------><
```

The getComboItems method returns the number of items in the combo box.

Arguments:

The only argument is:

id
: The ID of the combo box

3.13.9. getCurrentComboIndex

```
>>--getCurrentComboIndex(--id--)-----------------><
```

The getCurrentComboIndex method returns the index of the currently selected item within the list. See *getComboBoxData* for information on how to retrieve the selected combo box item.

Arguments:

The only argument is:

id

The ID of the combo box.

Example:

The following example displays the line number of the currently selected combo box item within entry line 240:

```
::class MyListDialog subclass UserDialog
     .
     .
     .
::method init
   self~Init:super
   self~ConnectList(230, "ListSelected")
     .
     .
     .
::method ListSelected
   line = self~getCurrentComboIndex(230)
   setEditData(240, line)
```

Method **`ListSelected`** is called each time the selected item within the combo box changes.

3.13.10. insertComboEntry

```
>>--insertComboEntry(--id--,--+-------+--,--string--)--------------------------><
                              +-index-+
```

The insertComboEntry method inserts a string into the list of a combo box.

Arguments:

The arguments are:

id

The ID of the combo box.

index

The index (line number) where you want to insert the new item. If this argument is omitted, the new item is inserted after the currently selected item.

string

The data string to be inserted.

Example:

This statement inserts **`The new third line`** after the second line into the list of combo box 103:

```
MyDialog~insertComboEntry(103, 2, "The new third line")
```

3.13.11. setCurrentComboIndex

```
>>--setCurrentComboIndex(--id--+----------+--)---><
                               +-,--index-+
```

The setCurrentComboIndex method selects the item with the given index within the list. If called without an index, all items in the list are deselected. See *setComboBoxData* for information on how to select a combo box item using a data value.

Arguments:

The arguments are:

id

The ID of the combo box.

index

The index within the combo box.

3.14. Data Attribute Methods

3.14.1. Understanding Data Attributes

This section contains methods for working with **data attributes**. It contains methods to create a connection between an underlying dialog control and an attribute of the Rexx dialog object, methods for **getting** and **setting** the **data**, and methods to deal with **automatic data detection**. These data attributes are used to reflect the **values** associated with the underlying dialog controls. For instance, the value of an *Edit* control could be thought of as the text of the control.

In order to better understand the data attribute methods, it is useful to first discuss this concept in general. During the *original* development of ooDialog the abstraction used was that there were only two objects. One was the Rexx dialog object and the other was the *underlying* system dialog that the user sees on the screen. Dialog controls were not thought of as objects, but rather as the **data** of the underlying dialog. In this abstraction, the **value** of the data was the **state** of the dialog control.

For instance, a radio button can be either checked or not checked. Therefore, its value could be 0 (not checked) or 1 (checked.) Thus the data of the radio button would be a 1 or a 0. Using this abstraction of two objects, the Rexx dialog and the underlying on screen dialog, then leads to the idea of exchanging, or transferring, the data between the two objects. Or, typically, getting and setting the data of the underlying dialog. To facilitate this exchange of data, for each **data item**, (a dialog control,) in the underlying dialog an attribute is added to the Rexx dialog object. This attribute is then used to transfer data between the Rexx dialog object and the underlying dialog control. The value of each data attribute is a single string which represents the state of the dialog control.

The operation of **connecting** a dialog control is the action of creating the attribute (the data attribute) in the Rexx dialog object, and internally mapping the attribute to the specified dialog control in the underlying dialog. The operation of **setting data** then becomes the action of setting the state of the underlying dialog control to match the value of the corresponding attribute of the Rexx dialog object. The operation of **getting data** becomes the action of setting the value of the attribute in the Rexx dialog object to match the state of the corresponding dialog control in the underlying dialog.

The term *automatic data field detection* is used for this process of creating and connecting data attributes. In addition to creating and connecting the data attributes, if automatic data field detection is on, when the underling dialog is created, the state of its controls are automatically set to the values of the data attributes. Likewise, when the dialog is closed, the values of the data attributes in the Rexx dialog are set to the state of their corresponding controls in the underlying dialog.

Automatic data detection is implemented in all three main classes of dialogs, the *UserDialog*, *ResDialog*, and *RcDialog*, and of course their subclasses. Automatic data detection can be turned on or off. By default it is **on**. The best way to turn automatic data field detection off for a single dialog is to over-ride the *initAutoDetection* method in your subclass. The *.application*, an instance of the *ApplicationManager* class, can be used to change the default to **off** for the entire program.

Some dialog controls are not considered to be *data* and therefore data attributes are not created for those controls. (This is a quirk going back to the original implementation of of ooDialog.) The non-data controls are those, in general, that the user would not change or update in a dialog. Push *buttons*, *group* box and *static* controls are all considered non-data controls. There are no data attribute methods for these controls.

When automatic data detection is turned off, the programmer can use the *connnect data attriubte* methods to manually create and connect the data attribute for any control. However, with automatic detection off, the ooDialog framework will not set the state of the dialog controls to match the values of the connected attributes when the underlying dialog is created. Likewise, when the dialog ends the connected attributes of the Rexx dialog object are not updated to the state of the dialog controls. But, the programmer can use the *setData* and *getData* methods to do this herself.

The implementation of automatic data detection in the three main classes of dialogs is slightly different in each class due to the differences in the classes:

UserDialog:

In a *UserDialog* the *connnect data attriubte* methods are called automatically from the *create...* methods of the **`UserDialog`**. Each of the create... methods takes an optional argument that allows the programmer to specify the data attribute name. If that argument is omitted, the attribute name is *automatically* generated. This behavior is only when automatic data detection is on. When it is off, no data connection is made, even if the attribute name is specified.

RcDialog

A *RcDialog* is actually a subclass of the **`UserDialog`**. Therefore, when automatic data detection is on, the connect... attribute methods are invoked automatically during the creation of each dialog control. The difference is that there is no option to specify the attribute name, the name is **always** generated automatically.

ResDialog

With a *ResDialog* the data connections are made after the underlying dialog is created. Internally, the numeric resource ID and text of each control in the dialog is determined. Then, a data attribute for each control is created and connected. Again, there is no way for the programmer to specify the data attribute name, it is always automatically constructed.

Obviously, the data attribute itself can not be accessed until it is created. For each type of class this is at a subtly different point in the life cycle of the dialog.

In summary: When automatic data field detection is on, the ooDialog framework automatically creates an attribute in the dialog object for each dialog control in the underlying dialog. The attribute is connected to the control through an internal mapping maintained by the framework. Once the connection is made, it can be used to get or set the *data* of the control. The data of a control is a string representation of the control's state. When each underlying dialog is created, the state of all its controls is set to the value of its connected data attribute. When the underlying dialog is closed by the user, the process is reversed. Each data attribute is updated to reflect the state of the dialog control when the dialog was closed. For the most part, the name of the data attribute is generated automatically by the framework. When automatic data field detection is off, none of this is done, but there are methods the programmer can use to manually do the same things.

3.14.2. Data Attribute Names

Except when the programmer explicitly names the data attribute in one of the *connnect data attriubte* methods, or one of the *create...* methods of the **`UserDialog`**, the data attribute names are generated internally. Even when the name is explicitly specified by the programmer, if the name is not a valid method name or is a duplicate method name, the attribute name is generated internally. The rules for how the attribute names are generated can be a little difficult to grasp. The following list is intended to be complete description of the rules in all circumstances.

connect... data attribute methods:

When the programmer uses one of the *connnect data attriubte* methods, the attribute name can be omitted, or the name specified could not be valid. For these cases, the internally generated name is created in this way:

If the name is omitted, then the name is created by prepending **`DATA`** with the resource ID argument. Recall that the resource ID argument can possibly be a symbolic ID. Example:

```
-- The attribute name will be DATA202
self~connectEdit(202)

-- The attribute name will be DATAIDC_EDIT_FNAME
self~connectEdit(IDC_EDIT_FNAME)
```

Whether the attribute name was specified or not, before the attribute is actually added to the dialog object, the name is checked to see that it is a valid method name, that it is not the empty string, and not a duplicate of an already existing method name in the dialog. When the check fails, the name is created by prepending **`DATA`** with the **numeric** value of the resource ID argument. This is true even when the id argument is specified as a symbolic ID. Example:

```
self~constDir[IDC_EDIT_FNAME] = 344

-- The attribute name will be DATA202
self~connectEdit(202, "COPY")

-- The attribute name will be DATA344
self~connectEdit(IDC_EDIT_FNAME, 'FNAME+ADDRESSBOOK')
```

UserDialog create... methods:

In the typical **`UserDialog`** all controls are added to the dialog by using one of the *create...* methods. This is similar to using one of the connect... methods. The attribute name argument can be omitted, or the specified attribute name might not be valid. For these cases, the internally generated name is created in this way:

If the name is omitted, and the dialog control is **not** a check box or radio button, then the name is created by prepending **`DATA`** with the numeric value of the resource ID argument. Example:

```
self~constDir[IDC_EDIT_FNAME] = 344

-- The attribute name will be DATA344
self~createEdit(344, 10, 10, 100, 12, "AUTOSCROLLH MULTILINE")

-- The attribute name will be DATA344
self~createEdit(IDC_EDIT_FNAME, 10, 10, 100, 12, "AUTOSCROLLH MULTILINE")
```

If the name is omitted and the dialog control **is** a check box or radio button, then the name is created from the text of the **label** of the button. First all spaces, ampersands, and colons, if any, are removed from the label. The result is used for the attribute name. Examples:

```
self~constDir[IDC_CHK_LN] = 200
self~constDir[IDC_CHK_MAIL] = 201
```

```
self~constDir[IDC_CHK_RECEIPT] = 202

-- The attribute name will be USELASTNAME
self~createCheckBox(IDC_CHK_LN, 10, 10, 80, 12, "GROUP", "&Use last name")

-- The attribute name will be MAILITEM
self~createCheckBox(IDC_CHK_MAIL, 10, 27, 80, 12, "RBUTTON RIGHT", "Mail item:")

-- The attribute name will be RECEIPT
self~createCheckBox(IDC_CHK_RECEIPT, 10, 45, 80, 12, , "Receipt")
```

Whether the attribute name was specified or generated by the framework, the name is always checked to see if it is the empty string, not a valid method name, or a duplicate of an already existing method name in the dialog. When the check fails, then the name is created by prepending **DATA** with the **numeric** value of the resource ID argument. This is true even when the id argument is specified as a symbolic ID. Examples:

```
self~constDir[IDC_EDIT_CLASS] = 344
self~constDir[IDC_CHK_DEFAULTNAME] = 201
self~constDir[IDC_CHK_COPY] = 202

-- The attribute name will be DATA344 (CLASS is an existing method name.)
self~createEdit(IDC_EDIT_CLASS, 10, 10, 100, 12, "AUTOSCROLLH MULTILINE", "Class")

-- The attribute name will be DATA201 (defaultName is an existing method name.)
self~createCheckBox(IDC_CHK_DEFAULTNAME, 10, 27, 80, 12, , "Default Name")

-- The attribute name will be DATA202
self~createCheckBox(IDC_CHK_COPY, 10, 45, 80, 12, , "&Copy")
```

ResDialog:

For a **ResDialog**, the attribute names are always generated by the ooDialog framework, there is no way for the programmer to specify the name. When the underlying dialog is created, the numeric resource ID and the text of all the dialog controls is collected. For **each** control, the data attribute is created using the **text** of the control.

Note this important detail: every dialog control can have text associated with it when compiled from a resource *script*. Although for many controls, text is not usually associated with them, it is always possible that they do have text.

The text of the control has all spaces, ampersands, and colons removed. The result is checked as usual. If it is the empty string, or not a valid method name, or a duplicate of an existing method name, then the attribute name is changed to **DATA** prepended to the resource ID. The attribute name then follows the pattern(s) already seen. Examples:

```
-- For an edit control with resource ID 344 and no associated text:
DATA344

-- For a check box with resource ID 400 and a label of "Compile":
COMPILE

-- For a radio button with resource ID 200 and a label of "&Copy":
DATA200

-- For a tree view with resource 320 and text associated of "C: Drive"
CDRIVE

-- Same tree view with no text assoicate, (the usual case):
DATA320

-- etc..
```

RcDialog:

Although a **`RcDialog`** is actually a subclass of a **`UserDialog`** and the dialog controls are added to the dialog *template* through the *create...* methods, the generated attribute names have a different pattern. If the resource ID of the control in the resource *script* file is a symbolic ID, then the attribute name will be the symbolic ID. On the other hand, if the resource ID is numeric in the resource script file, then the attribute name will be **`DATA`** prepended to the numeric ID. Example:

```
  -- For this resource script file:

//
// Dialog resources
//
LANGUAGE LANG_NEUTRAL, SUBLANG_NEUTRAL
IDD_DIALOG1 DIALOGEX 0, 0, 237, 131
STYLE  DS_MODALFRAME | WS_VISIBLE | WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_SYSMENU
CAPTION "Dialog"
FONT 8, "MS Shell Dlg 2", 400, 0, 1
{
    EDITTEXT        1003, 27, 22, 116, 47, WS_GROUP | ES_AUTOHSCROLL | ES_MULTILINE
    EDITTEXT        IDC_EDIT2, 27, 84, 116, 12, ES_AUTOHSCROLL
    AUTOCHECKBOX    "Hello There", IDC_CHK_TEST, 27, 109, 52, 10
    DEFPUSHBUTTON   "OK", IDOK, 125, 107, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 177, 107, 50, 14
    LTEXT           "Just a test of a static control", IDC_ST_TEST, 163, 28, 54, 24,
 SS_LEFT
}

  -- The attribute names will be:

  DATA1003
  IDC_EDIT2
  IDC_CHK_TEST
```

However, as always, the attribute name is checked to be a valid method name, not the empty string, and not a duplicate of an existing method name. If the resource script file is valid, it can not be the empty string. Usually, programmers do not name their symbolic IDs in a fashion that would conflict with an existing method name, but there is no reason why they could not. If the check fails, then the attribute name will once again be **`DATA`** prepended to the **numeric** value of the resource ID. Example:

```
  -- For this resource script file:

#define COPY 1004

//
// Dialog resources
//
LANGUAGE LANG_NEUTRAL, SUBLANG_NEUTRAL
IDD_DIALOG1 DIALOGEX 0, 0, 237, 131
STYLE  DS_MODALFRAME | WS_VISIBLE | WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_SYSMENU
CAPTION "Dialog"
FONT 8, "MS Shell Dlg 2", 400, 0, 1
{
    EDITTEXT        1003, 27, 22, 116, 47, WS_GROUP | ES_AUTOHSCROLL | ES_MULTILINE
    EDITTEXT        IDC_EDIT2, 27, 84, 116, 12, ES_AUTOHSCROLL
    AUTOCHECKBOX    "Hello There", COPY, 27, 109, 52, 10
    DEFPUSHBUTTON   "OK", IDOK, 125, 107, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 177, 107, 50, 14
    LTEXT           "Just a test of a static control", IDC_ST_TEST, 163, 28, 54, 24,
 SS_LEFT
}

  -- The attribute names will be:
```

```
DATA1003
IDC_EDIT2
DATA1004
```

3.14.3. Problems with Data Attributes

In the author's mind, there are a number of problems with the whole data attribute concept. In no particular order, some of them are brought up here.

The abstraction that there are only two objects, the Rexx dialog object and the underlying operating system dialog, is a little outdated. In this abstraction, the state of a dialog control must be transferred back and forth through the dialog object. In addition, the original users of the abstraction believed the *data* of a dialog control could only be represented by a single string. The abstraction works somewhat for simple controls like a check box, a radio button, or an edit control. However it quickly starts to break down with more complex controls like tree views and list views. Indeed, when the more complicated controls were added to ooDialog, there were no get and set data methods implemented for them. There is no *getTreeViewData* or *setListViewData* methods.

A more flexible abstraction is that everything is an object, that all the dialog controls are objects. In this abstraction, the state of a dialog control is set or queried directly through methods of the dialog control object. Current development in ooDialog is done using this abstraction.

Presumably the automatic data field detection was done to make things easier for the Rexx programmer. However, the whole process is actually quite confusing. The data attributes themselves are not visible in the program, which often makes it difficult to read and understand a program. When the attribute names are generated automatically, it is often quite difficult to know exactly what the name is.

With auto detection on, the states of the dialog controls are set to the data (values) of the corresponding data attributes. This is done **after** *initDialog* is invoked. The consequence of this is that, if the programmer explicitly sets the state of the dialog controls in the *initDialog*() method, the ooDialog framwork will then **reset** the state of all the dialog controls afterwards. This can be disconcerting to the programmer if automatic data field detection is not understood.

3.14.4. autoDetection

```
>>--autoDetection--------------------------------><
```

When the *autoDetection* method is invoked, it switches on *automatic data detections [156]*.

Arguments:

This method has no arguments.

Return value:

This method does not return anything.

3.14.5. Connect Data Attribute Methods

The methods in this section are used to manually create and connect a data *attribute*. Their purpose is to manually do what is done when *automatic* data field detection is on. They should only be used when automatic detection is off.

3.14.5.1. connectCheckBox

```
>>--connectCheckBox(--id--+-------------------+--)---------------><
                          +-,--attributeName--+
```

The *connectCheckBox* method connects a check box in the underlying dialog, through its resource ID, to a newly created data *attribute* in the Rexx dialog object.

Arguments:

The arguments are:

id [required]

The resource ID of the check box control to be connected. Can be numeric or symbolic.

attributeName [optional]

The name of the data attribute to be created. If this argument is omitted or invalid, the attribute name is *automatically*/>.

Return value:

The possible return codes are:

-1

A symbolic ID was used and it could not be resolved.

0

No error.

1

The data attribute was created but it was not connected. (The data connection table is full.)

Remarks:

The attribute has to be synchronized with the check box control manually. This can be done globally with the *setData* and *getData* methods. For a finer grain of control use the *setCheckBoxData* or *getCheckBoxData* methods. The *setDataAttribute* or *getDataAttribute* methods could also be used.

The *data* of a check *CheckBox* is defined to be its check state. For normal check boxes this is 0 for not checked and 1 for checked. Three-state check boxes have the additional indeterminate state. The value for the indeterminate state is 2.

The *connectCheckBox* method should only be used when *automatic data detections [156]* is off.

Example:

The *example* for the *connectEdit* method demonstrates all the pertinent steps to manually connect and use a data attribute for a dialog control. Simply substitute a check box for the edit control.

3.14.5.2. connectComboBox

```
>>--connectComboBox(--id--+------------------+--)---------------><
                          +-,-attributeName--+
```

The *connectComboBox* method creates a data *attribute* in the Rexx dialog object and connects it to a combo box control in the underlying dialog using its resource id.

Arguments:

id [required]

The resource ID of the combo box control to be connected. Can be numeric or symbolic.

attributeName [optional]

The name of the data attribute to be created. If this argument is omitted or invalid, the attribute name is *automatically*.

Return value:

The possible return codes are:

-1

A symbolic ID was used and it could not be resolved.

0

No error.

1

The data attribute was created but it was not connected. (The data connection table is full.)

Remarks:

The attribute has to be synchronized with the combo box control manually. This can be done globally with the *setData* and *getData* methods. For a finer grain of control use the *setComboBoxData* or *getComboBoxData*) methods. The *setDataAttribute* or *getDataAttribute* methods can also be used.

The *data* of a combo *ComboBox* is defined to be the text in the selection field or the selected combo box item.

The *connectComboBox* should only be used when *automatic data detections [156]* is off.

Example:

The *example* for the *connectEdit* method demonstrates all the pertinent steps to manually connect and use a data attribute for a dialog control. Simply substitute a combo box for the edit control.

3.14.5.3. connectDateTimePicker

```
>>--connectDateTimePicker(--id--+------------------+--)---------><
                                +-,--attributeName-+
```

The *connectDateTimePicker* method connects a date time picker control in the underlying dialog, through its resource ID, to a newly created data *attribute* in the Rexx dialog object.

Arguments:

The arguments are:

id [required]

The resource ID of the date time picker control to be connected. Can be numeric or symbolic.

attributeName [optional]

The name of the data attribute to be created. If this argument is omitted or invalid, the attribute name is *generated automatically*.

Return value:

The possible return codes are:

-1
: A symbolic ID was used and it could not be resolved.

0
: No error.

1
: The data attribute was created but it was not connected. (The data connection table is full.)

Remarks:

The attribute has to be synchronized with the date time control manually. This can be done globally with the *setData* and *getData* methods. For a finer grain of control use the *setDataAttribute* or *getDataAttribute* methods.

The *data* of a date time *DateTimePicker* is defined to be a **`DateTime`** object that represents it currently selected date and time.

The *connectDateTimePicker* method should only be used when *automatic data detections [156]* is off.

Example:

The *example* for the *connectEdit* method demonstrates all the pertinent steps to manually connect and use a data attribute for a dialog control. Simply substitute a date time picker for the edit control.

3.14.5.4. connectEdit

```
>>--connectEdit(--id--+-----------------+--)----><
                      +-,--attributeName-+
```

The *connectEdit* method creates a new data *attribute* attribute in the Rexx dialog object and connects it to the underlying edit control using its resource id.

Arguments:

The arguments are:

id [required]
: The resource ID of the edit control to be connected. Can be numeric or symbolic.

attributeName [optional]
: The name of the data attribute to be created. If this argument is omitted or invalid, the attribute name is *generated automatically*.

Return value:

The possible return codes are:

-1
: A symbolic ID was used and it could not be resolved.

0
: No error.

1
: The data attribute was created but it was not connected. (The data connection table is full.)

Remarks:

The attribute has to be synchronized with the edit control manually. This can be done globally with the *setData* and *getData* methods. For a finer grain of control use the *setEditData* or *getEditData* methods. The *setDataAttribute* or *getDataAttribute* methods can also be used.

The *data* of an edit control is defined to be the text entered in the control. If there is no text, the data is the empty string.

The *connectEdit* should only be used when *automatic data detections [156]* is off.

Example:

In the following example, the edit control with symbolic resource ID IDC_EDIT is connected to the Rexx dialog object's *NAME* attribute. The connection process creates the data attribute. The string: *Put your name here* is assigned to the newly created data attribute. When the dialog is executed, the underlying edit control is manually synchronized with the attribute. This has the effect of setting the text of the edit control to the value of the attribute, which is the string, Put your name here.

When the dialog is closed by the user with an *ok* command, the data attribute is manually synchronized with the state of the edit control. The state of an edit control is defined to be the text within the control. This synchronization has the effect of copying whatever text is in the edit control to the data attribute. Then, when the dialog has terminated, the name the user entered is displayed. (Of course the user may not have entered her name. What is displayed is whatever the text in the edit control was when the dialog was closed.)

```
  dlg = .NameDlg~new("nameExample.dll", IDD_DIALOG1, , "nameExample.h" )

  if dlg~initCode == 0 then do
    dlg~noAutoDetection
    dlg~connectEdit(IDC_EDIT, "NAME")
    dlg~name = "Put your name here"

    ret = dlg~execute("SHOWTOP", IDI_DLG_OOREXX)

    if ret == 1 then say "The user's name is:" dlg~name
    else say "The user canceled."
  end

return 0

::requires "ooDialog.cls"

::class 'NameDlg' subclass ResDialog

::method initDialog
  self~setDataAttribute("NAME")

::method ok
  self~getDataAttribute("NAME")
  self~ok:super
```

3.14.5.5. connectListBox

```
>>--connectListBox(--id--+-------------------+--)--------------><
                         +-,--attributeName--+
```

The *connectListBox* method uses the resource ID of the control to connect a list box in the underlying dialog to a newly created data *attribute* in the Rexx dialog object.

Arguments:

The arguments are:

id [required]

The resource ID of the list box to be connected. Can be numeric or symbolic.

attributeName [optional]

The name of the data attribute to be created. If this argument is omitted or invalid, the attribute name is *generated automatically*.

Return value:

The possible return codes are:

-1

A symbolic ID was used and it could not be resolved.

0

No error.

1

The data attribute was created but it was not connected. (The data connection table is full.)

Remarks:

The attribute has to be synchronized with the list box manually. This can be done globally with the *setData* and *getData* methods. For a finer grain of control use the *setListBoxData* or the *getListBoxData*. The *setDataAttribute* or *getDataAttribute* methods could also be used.

The *data* of a **single-selection** *ListBox* is defined to be text of the selected item in the list box.

The *data* of a **multi-selection** *ListBox* is defined to be a string containing the item numbers of its selected items, separated by a blank. For a multi-selection list box the string could contain any number of list items depending on what the user has selected, (for getting data,) or what the programmer is setting to be selected, (for setting data.)

The *connectListBox* method should only be used when *automatic data detections [156]* is off.

Example Single Selection:

This example shows how to use the *connectListBox* method with a single-selection list box. It is very similar to the *example* for the *connectEdit* method. The code can be copied and pasted into the appropriate files to create a working example.

```
/* singleSelection.rex */

  dlg = .SingleSelectionDlg~new("singleSelection.rc", IDD_SINGLE_SELECTION_DLG, ,
 "singleSelection.h" )
  if dlg~initCode == 0 then do
    dlg~noAutoDetection
    dlg~connectListBox(IDC_LB_STATES, "LISTBOX")
    dlg~listBox = "California"

    ret = dlg~execute("SHOWTOP", IDI_DLG_OOREXX)

    if ret == 1 then say "The user's home state is:" dlg~listBox
    else say "The user canceled."
  end

return 0

::requires "ooDialog.cls"

::class 'SingleSelectionDlg' subclass RcDialog
```

```
::method initDialog

  states = self~getStateArray
  listBox = self~newListBox(IDC_LB_STATES)

  do state over states
    listBox~add(state)
  end

  self~setDataAttribute("LISTBOX")

::method ok
  self~getDataAttribute("LISTBOX")
  self~ok:super


::method getStateArray private

  a = .array~of(...)
  return a

/* End of singleSelection.rex */


/* Place this code in a file named "singleSelection.h" */

#ifndef IDC_STATIC
#define IDC_STATIC (-1)
#endif

#define IDD_SINGLE_SELECTION_DLG                 100
#define IDC_LB_STATES                           1003

/* Place this code in a file called singleSelection.rc. Two lines will wrap below:
 * The line that begins "STYLE DS_3DLOOK" is all one line until the line that
 * begins with "CAPTION" Likewise the "LISTBOX" line is all one line until the
 * "DEFPUSHBUTTON" line. These lines will need to be fixed after the copy and
 * paste.
 */


#include <windows.h>
#include "singleSelection.h"

LANGUAGE 0, SUBLANG_NEUTRAL
IDD_SINGLE_SELECTION_DLG DIALOG 0, 0, 186, 148
STYLE DS_3DLOOK | DS_CENTER | DS_MODALFRAME | DS_FIXEDSYS | WS_VISIBLE | WS_BORDER |
 WS_CAPTION | WS_DLGFRAME | WS_POPUP | WS_SYSMENU
CAPTION "Home State"
FONT 8, "Ms Shell Dlg 2"
{
    CTEXT           "Select Your Home State", IDC_STATIC, 29, 12, 127, 11, SS_CENTER
    LISTBOX         IDC_LB_STATES, 25, 31, 137, 84, WS_TABSTOP | WS_VSCROLL |
 LBS_NOINTEGRALHEIGHT | LBS_SORT | LBS_NOTIFY
    DEFPUSHBUTTON   "OK", IDOK, 71, 124, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 126, 124, 50, 14
}

/* Use this code to replace the .array~of(...) statement in the Rexx program. */
  a = .array~of( -
                "Alabama", -
                "Alaska", -
                "American Samoa", -
                "Arizona", -
                "Arkansas", -
                "California", -
```

```
                 "Colorado", -
                 "Connecticut", -
                 "Delaware", -
                 "District of Columbia", -
                 "Florida", -
                 "Georgia", -
                 "Guam", -
                 "Hawaii", -
                 "Idaho", -
                 "Illinois", -
                 "Indiana", -
                 "Iowa", -
                 "Kansas", -
                 "Kentucky", -
                 "Louisiana", -
                 "Maine", -
                 "Maryland", -
                 "Massachusetts", -
                 "Michigan", -
                 "Minnesota", -
                 "Mississippi", -
                 "Missouri", -
                 "Montana", -
                 "Nebraska", -
                 "Nevada", -
                 "New Hampshire", -
                 "New Jersey", -
                 "New Mexico", -
                 "New York", -
                 "North Carolina", -
                 "North Dakota", -
                 "Northern Marianas Islands", -
                 "Ohio", -
                 "Oklahoma", -
                 "Oregon", -
                 "Pennsylvania", -
                 "Puerto Rico", -
                 "Rhode Island", -
                 "South Carolina", -
                 "South Dakota", -
                 "Tennessee", -
                 "Texas", -
                 "Utah", -
                 "Vermont", -
                 "Virginia", -
                 "Virgin Islands", -
                 "Washington", -
                 "West Virginia", -
                 "Wisconsin", -
                 "Wyoming")
```

Example Multi Selection:

This example shows how to use the *connectListBox* method with a multi-selection list box. It is also very similar to the *example* for the *connectEdit* method.

```
/* multiSelection.rex */

  dlg = .MultiSelectionDlg~new("multiSelection.rc", IDD_MULTI_SELECTION_DLG, ,
 "multiSelection.h" )
  if dlg~initCode == 0 then do
    dlg~noAutoDetection
    dlg~connectListBox(IDC_LB_STATES, "LISTBOX")
    dlg~listBox = "4 6 7 49"

    ret = dlg~execute("SHOWTOP", IDI_DLG_OOREXX)
    if ret == 1 then do
      indexes = dlg~listBox~makeArray(" ")
      states = getStateArray()

      say "The user has visited these states:"
      do index over indexes
        say " " states[index]
      end
    end
    else do
      say "The user canceled."
    end
  end

return 0

::requires "ooDialog.cls"

::class 'MultiSelectionDlg' subclass RcDialog

::method initDialog

  states = getStateArray()
  listBox = self~newListBox(IDC_LB_STATES)

  do state over states
    listBox~add(state)
  end

  self~setDataAttribute("LISTBOX")

::method ok
  self~getDataAttribute("LISTBOX")
  self~ok:super


::routine getStateArray public
  a = .array~of(...)
  return a

/* Place this code in a file called multiSelection.rc. Two lines will wrap below:
 * The line that begins "STYLE DS_3DLOOK" is all one line until the line that
 * begins with "CAPTION" Likewise the "LISTBOX" line is all one line until the
 * "DEFPUSHBUTTON" line. These lines will need to be fixed after the copy and
 * paste.
 */

#include <windows.h>
#include "multiSelection.h"

LANGUAGE 0, SUBLANG_NEUTRAL
IDD_MULTI_SELECTION_DLG DIALOG 0, 0, 186, 148
STYLE DS_3DLOOK | DS_CENTER | DS_MODALFRAME | DS_FIXEDSYS | WS_VISIBLE | WS_BORDER |
 WS_CAPTION | WS_DLGFRAME | WS_POPUP | WS_SYSMENU
CAPTION "Visited States"
FONT 8, "Ms Shell Dlg 2"
{
    CTEXT           "Select the states you've visited", IDC_STATIC, 29, 12, 127, 11,
 SS_CENTER
    LISTBOX         IDC_LB_STATES, 25, 31, 137, 84, WS_TABSTOP | WS_VSCROLL |
 LBS_MULTIPLESEL | LBS_NOINTEGRALHEIGHT | LBS_SORT | LBS_NOTIFY
    DEFPUSHBUTTON   "OK", IDOK, 71, 124, 50, 14
    PUSHBUTTON      "Cancel", IDCANCEL, 126, 124, 50, 14
}
```

```
/* Place this code in a file called multiSelection.h */

/* Use this code to replace the .array~of(...) statement in the Rexx program. */
  a = .array~of( -
                "Alabama", -
                "Alaska", -
                "American Samoa", -
                "Arizona", -
                "Arkansas", -
                "California", -
                "Colorado", -
                "Connecticut", -
                "Delaware", -
                "District of Columbia", -
                "Florida", -
                "Georgia", -
                "Guam", -
                "Hawaii", -
                "Idaho", -
                "Illinois", -
                "Indiana", -
                "Iowa", -
                "Kansas", -
                "Kentucky", -
                "Louisiana", -
                "Maine", -
                "Maryland", -
                "Massachusetts", -
                "Michigan", -
                "Minnesota", -
                "Mississippi", -
                "Missouri", -
                "Montana", -
                "Nebraska", -
                "Nevada", -
                "New Hampshire", -
                "New Jersey", -
                "New Mexico", -
                "New York", -
                "North Carolina", -
                "North Dakota", -
                "Northern Marianas Islands", -
                "Ohio", -
                "Oklahoma", -
                "Oregon", -
                "Pennsylvania", -
                "Puerto Rico", -
                "Rhode Island", -
                "South Carolina", -
                "South Dakota", -
                "Tennessee", -
                "Texas", -
                "Utah", -
                "Vermont", -
                "Virginia", -
                "Virgin Islands", -
                "Washington", -
                "West Virginia", -
                "Wisconsin", -
                "Wyoming")
```

3.14.5.6. connectListView

```
>>--connectListView(--id--+------------------+--)---------------><
                          +-,--attributeName-+
```

The *connectListView* method connects a list view in the underlying dialog, through its resource ID, to a newly created data *attribute* in the Rexx dialog object.

Arguments:

The arguments are:

id [required]

The resource ID of the list view control to be connected. Can be numeric or symbolic.

attributeName [optional]

The name of the data attribute to be created. If this argument is omitted or invalid, the attribute name is *generated automatically*.

Return value:

The possible return codes are:

-1

A symbolic ID was used and it could not be resolved.

0

No error.

1

The data attribute was created but it was not connected. (The data connection table is full.)

Remarks:

The attribute has to be synchronized with the list view control manually. This can be done globally with the *setData* and *getData* methods. For a finer grain of control use the *setDataAttribute* or *getDataAttribute* methods.

The *data* of a list *ListView* is defined to be a string containing the blank separated index(es) of its selected item(s).

The *connectCheckBox* method should only be used when *automatic data detections [156]* is off.

Example:

The *example*/> for the *connectEdit* method demonstrates all the pertinent steps to manually connect and use a data attribute for a dialog control. Simply substitute a list view for the edit control.

3.14.5.7. connectMonthCalendar

```
>>--connectMonthCalendar(--id--+------------------+--)----------><
                               +-,--attributeName-+
```

The *connectMonthCalendar* method connects a month calendar control in the underlying dialog, through its resource ID, to a newly created data *attribute* in the Rexx dialog object.

Arguments:

The arguments are:

id [required]

The resource ID of the month calendar control to be connected. Can be numeric or symbolic.

attributeName [optional]

The name of the data attribute to be created. If this argument is omitted or invalid, the attribute name is *generated automatically*.

Return value:

The possible return codes are:

-1

A symbolic ID was used and it could not be resolved.

0

No error.

1

The data attribute was created but it was not connected. (The data connection table is full.)

Remarks:

The attribute has to be synchronized with the month calendar control manually. This can be done globally with the *setData* and *getData* methods. For a finer grain of control use the *setDataAttribute* or *getDataAttribute* methods.

The *data* of a *MonthCalendar* is defined to be a **`DateTime`** object that specifies the selected date in the month calendar control.

The *connectMonthCalendar* method should only be used when *automatic data detections [156]* is off.

Example:

The *example*/> for the *connectEdit* method demonstrates all the pertinent steps to manually connect and use a data attribute for a dialog control. Simply substitute a month calendar control for the edit control.

3.14.5.8. connectRadioButton

```
>>--connectRadioButton(--id--+-------------------+--)----------><
                             +-,--attributeName--+
```

The *connectRadioButton* method uses the resource id of a radio button control to connect the control in the underlying dialog to a newly created data*attribute* in the Rexx dialog object.

Arguments:

The arguments are:

id [required]

The resource ID of the radio button control to be connected. Can be numeric or symbolic.

attributeName [optional]

The name of the data attribute to be created. If this argument is omitted or invalid, the attribute name is *generated automatically*.

Return value:

The possible return codes are:

-1

A symbolic ID was used and it could not be resolved.

0
: No error.

1
: The data attribute was created but it was not connected. (The data connection table is full.)

Remarks:

The attribute has to be synchronized with the radio button control manually. This can be done globally with the *setData* and *getData* methods. For a finer grain of control use the *setRadioButtonData* or *getRadioButtonData* methods. The *setDataAttribute* or *getDataAttribute* methods can also be used.

The *data* of a *RadioButton* is defined to be the its check state. This is 0 for not checked and 1 for checked.

The *connectRadioButton* method should only be used when *automatic data detections [156]* is off.

Example:

The *example*/> for the *connectEdit* method demonstrates all the pertinent steps to manually connect and use a data attribute for a dialog control. Simply substitute a radio button control for the edit control.

3.14.5.9. connectTab

```
>>--connectTab(--id--+------------------+--)-----><
                     +-,--attributeName-+
```

The *connectTab* method uses the resource id of a tab control to connect the control in the underlying dialog to a newly created data *attribute* in the Rexx dialog object.

Arguments:

The arguments are:

id [required]
: The resource ID of the tab control to be connected. Can be numeric or symbolic.

attributeName [optional]
: The name of the data attribute to be created. If this argument is omitted or invalid, the attribute name is *generated automatically*.

Return value:

The possible return codes are:

-1
: A symbolic ID was used and it could not be resolved.

0
: No error.

1
: The data attribute was created but it was not connected. (The data connection table is full.)

Remarks:

The attribute has to be synchronized with the tab control manually. This can be done globally with the *setData* and *getData* methods. For a finer grain of control use the *setDataAttribute* or*getDataAttribute* methods.

The *data* of a *Tab* control is defined to the text of its currently selected tab.

The *connectTab* method should only be used when *automatic data detections [156]* is off.

Example:

The *example* for the *connectEdit* method demonstrates all the pertinent steps to manually connect and use a data attribute for a dialog control. Simply substitute a tab control for the edit control.

3.14.5.10. connectTrackBar

```
>>--connectTrackBar(--id--+-----------------+--)--------------><
                          +-,--attributeName-+
```

The *connectTrackBar* method uses the resource id of a trackbar control to connect the control in the underlying dialog to a newly created data *attribute* in the Rexx dialog object.

Arguments:

The arguments are:

id [required]

The resource ID of the trackbar control to be connected. Can be numeric or symbolic.

attributeName [optional]

The name of the data attribute to be created. If this argument is omitted or invalid, the attribute name is *generated automatically*.

Return value:

The possible return codes are:

-1

A symbolic ID was used and it could not be resolved.

0

No error.

1

The data attribute was created but it was not connected. (The data connection table is full.)

Remarks:

The attribute has to be synchronized with the trackbar control manually. This can be done globally with the *setData* and *getData* methods. For a finer grain of control use the *setDataAttribute* or *getDataAttribute* methods.

The *data* of a *Tab* control is defined to be the current logical position of the slider in the trackbar.

The *connectTab* method should only be used when *automatic data detections [156]* is off.

Example:

The *example* for the *connectEdit* method demonstrates all the pertinent steps to manually connect and use a data attribute for a dialog control. Simply substitute a trackbar control for the edit control.

3.14.5.11. connectTreeView

```
>>--connectTreeView(--id--+------------------+--)---------------><
                          +-,--attributeName-+
```

The *connectTreeView* method creates a data *attribute* in the Rexx dialog object and connects it to a tree view control in the underlying dialog using its resource id.

Arguments:

id [required]

The resource ID of the tree view control to be connected. Can be numeric or symbolic.

attributeName [optional]

The name of the data attribute to be created. If this argument is omitted or invalid, the attribute name is *generated automatically*.

Return value:

The possible return codes are:

-1

A symbolic ID was used and it could not be resolved.

0

No error.

1

The data attribute was created but it was not connected. (The data connection table is full.)

Remarks:

The attribute has to be synchronized with the tree view control manually. This can be done globally with the *setData* and *getData* methods. For a finer grain of control use the *setDataAttribute* or *getDataAttribute* methods.

The *data* of a tree *TreeView* is defined to be the text of the selected tree view item.

The *connectTreeview* should only be used when *automatic data detections [156]* is off.

Example:

The *example*/> for the *connectEdit* method demonstrates all the pertinent steps to manually connect and use a data attribute for a dialog control. Simply substitute a tree view for the edit control.

3.14.5.12. connectUpDown

```
>>--connectUpDown(--id--+------------------+--)--><
                        +-,--attributeName-+
```

The *connectUpDown* method connects an up down control in the underlying dialog, through its resource ID, to a newly created data *attribute* in the Rexx dialog object.

Arguments:

The arguments are:

id [required]

The resource ID of the up down control to be connected. Can be numeric or symbolic.

attributeName [optional]

The name of the data attribute to be created. If this argument is omitted or invalid, the attribute name is *generated automatically*.

Return value:

The possible return codes are:

-1

A symbolic ID was used and it could not be resolved.

0

No error.

1

The data attribute was created but it was not connected. (The data connection table is full.)

Remarks:

The attribute has to be synchronized with the up down control manually. This can be done globally with the *setData* and *getData* methods. For a finer grain of control use the *setDataAttribute* or *getDataAttribute* methods.

The *data* of a up *UpDown* control is defined to be the value of its current position.

The *connectUpDown* method should only be used when *automatic data detections [156]* is off.

Example:

The *example* for the *connectEdit* method demonstrates all the pertinent steps to manually connect and use a data attribute for a dialog control. Simply substitute an up down control for the edit control.

3.14.6. getCheckBoxData

```
>>--getCheckBoxData(--id--)----------------------><
```

The *getCheckBoxData* method gets the data *attribute* of a check box.

Arguments:

The only argument is:

id

The resource ID of the check box. May be numeric or symbolic.

Return value:

The *data* of a check box is considered to be whether it is checked or not. If the check box is checked the return is 1 and if unchecked the return is 0.

Remarks:

The check box **should** have been previously connected to a data attribute. If it has not, or if the resource ID is not the ID of a check box, the results are unpredictable.

Details

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

3.14.7. getComboBoxData

```
>>--getComboBoxData(--id--)----------------------><
```

The *getComboBoxData* method gets the data *attribute* of a combo box.

Arguments:

The only argument is:

id [required]

The resource ID of the combo box. May be numeric or symbolic.

Return value:

The *data* of a combo box is considered to be the text of the selection field in the combo *ComboBox*. The current text in the selection field is returned, which may be the empty string if there is no text in the selection field.

Remarks:

The combo box **should** have been previously connected to a data attribute. If it has n0t, or if the resource ID is not the ID of a combo box, the results are unpredictable.

For drop down list combo boxes, the text in the selection field is the same as the text of the current selection. The text returned may still be the empty string if there is no current selection.

However, for simple and drop down combo boxes things are not so simple. The text in the selection field could be text the user typed in and there may be no current selection. Or, for these types of combo boxes, there might be a current selection, but the user may have edited the text in the selection field so that it no longer matches the text of the current selection.

To get the index of the current selection, use the *getCurrentComboIndex* method.

Details

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

3.14.8. getControlData

```
>>--getControlData(--id--)-----------------------><
```

The *getControlData* method gets the data *attribute* of a dialog control, regardless of the type of the control. The control **should** have been previously connected to a data attribute.

Arguments:

The arguments are:

id [required]

The resource ID of the dialog control. May be symbolic or numeric.

Return Value:

The return will be the *value* of the control data. This is dependent on the type of the control. For an edit control it will be the text entered in the control, for a radio button it will be 0 or 1, depending on if the radio button is unchecked or checked, for a tree view it will be the text of the selected item in the tree, etc.. The defined value for the *data* of any dialog control is documented in the *connect data attriubte* method for the control.

Remarks:

If the control has not been previously connected to a data attribute, the implementation of this method assumes the control is an edit control and returns the text of the control. Since all windows have text value, which may be the empty string, this is okay. However this is not the intent of this method. The method is intended to only be used with controls connected to a data attribute.

It appears that the original intent of this method was to be used for controls such as tree views, list views, tabs, etc.. When these controls were added to ooDialog, no corresponding getTreeViewData(), getProgressBarData(), etc., methods were implemented. The generic *getControlData*() method works for all connected controls, but it works best if the programmer knows the type of control.

A better method to use, if the programmer is insistent on thinking in terms of data attributes, is probably to use the *data* method of the dialog *control* object.

Details

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

3.14.9. getData

```
>>--getData--------------------------------------><
```

The *getData* method sets the Rexx dialog data *attribute* values to the state of the underlying, connected, dialog controls.

Arguments:

There are no arguments for this method.

Return value:

0 on success or 1 on error.

Remarks

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

Details

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

Example:

The following example continues the *setData* example:

```
     .
     .
     .
dialog~connectEdit(102, "EDIT_1")
dialog~connectCheckBox(201, )
dialog~ConnectListbox(203, "LISTBOX_DAYS")
     .
     .
     .
/* process the dialog */
     .
     .
     .
```

```
dialog~getData       /* set data attributes to the state of the dialog controls */
say dialog~EDIT_1
say dialog~DATA201
say dialog~LISTBOX_DAYS
```

3.14.10. getDataAttribute

```
>>--getDataAttribute(--attributeName--)---------->< 
```

The *getDataAttribute* method transfers the data of a dialog control to the connected data *attribute* of the Rexx dialog object.

Arguments:

The only argument is:

attributeName [required]

The name of the data attribute that receives the data of the underlying dialog control.

Remarks:

The terms *transfers* and *receives* the data are holdovers from the original documentation and the original data *attribute* abstraction. What the *getDataAttribute* method actually does is to set the value of the connected data attribute to the defined value of the underlying dialog control. The defined value for the *data* of any dialog control is documented in the *connnect data attriubte* method for the control.

Details

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

Example:

The following example shows how to get the data of a dialog control and synchronize its connected data attribute.

```
self~getDataAttribute("FIRSTNAME")
if self~firstName \= "" then ...
```

3.14.11. getDataStem

```
>>--getDataStem(--dataStem.--)-------------------><
```

The *getDataStem* method is the reverse of the *setDataStem* method. The method gets the *data* of all the dialog controls in the underlying dialog that have been connected to a data *attribute* of the Rexx dialog object. The data of each control is copied to the specified stem.

A stem index for each connected control is set. The stem index is the numeric ID for the control. The value at that index is the data of the matching dialog control. In addition, if the *constDir* directory has a symbolic ID for the numeric ID of the control, a stem index of the symbolic ID is also set. This of course only works reliably if the Rexx programmer uses unique numeric IDs for all resources.

Arguments:

dataStem. [required]

The name of a stem variable in which the data is returned. Remember the trailing period.

Return value:

Returns 0, always.

Details

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

3.14.12. getEditData

```
>>-aBaseDialog~getEditData(--id--)----------------------------><
```

The *getEditData* method gets the data *attribute* of an edit control.

Arguments:

The only argument is:

id [required]

The resource ID of the edit control. May be numeric or symbolic.

Return value:

The *data* of an edit control is considered to the text entered in the control. This method returns the text in the edit control, which may be the empty string if the user has not entered any text.

Remarks:

The edit control **should** have been previously connected to a data attribute. If it has not, or if the resource ID is not the ID of a edit control, the results are unpredictable.

Details

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

3.14.13. getListBoxData

```
>>--getListBoxData(--id--)------------------------><
```

The *getListBoxData* method gets the data *attribute* of a list box.

Arguments:

The only argument is:

id [required]

The resource ID of the list box. May be numeric or symbolic.

Return value:

For a single-selection list box, the *data* of the list box is considered to be the text of the selected item. The text of the selected item is returned, or the empty string if no item is selected.

However, for a multi-selection list box, the *data* is considered to be the index numbers of the selected items. The selected indexes are returned in a string with the indexes separated by spaces. If no item is selected then the empty string is returned.

Remarks:

The list box **should** have been previously connected to a data attribute. If it has not, or if the resource ID is not the ID of a list box, the results are unpredictable.

For a multi-selection list box, the string is restricted to 1500 indexes.

Details

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

Example:

The following example shows how to handle a multiple-selection list box. It parses the returned string as long as it contains an index.

```
selLines = MyDialog~getListBoxData(555)
do until selLines = ""
   parse var selLines aLine selLines
   say aLine
end
```

3.14.14. getRadioButtonData

```
>>--getRadioButtonData(--id--)-------------------><
```

The *getRadioButtonData* method gets the data *attribute* of a a radio button.

Arguments:

The only argument is:

id

The resource ID of the radio button. May be numeric or symbolic.

Return value:

The *data* of a radio button is considered to be whether it is checked or not. If it is checked the return is 1 and if unchecked the return is 0.

Remarks:

The radio button **should** have been previously connected to a data attribute. If it hasn't, or if the resource ID is not the ID of a radio button, the results are unpredictable.

Details

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

3.14.15. initAutoDetection

```
>>-initAutoDetection---------------------------------><
```

The *initAutoDetection* method is invoked automatically by the ooDialog framework when any dialog is first instantiated. The default implementation of *initAutoDetection* turns *automatic* data detection on.

Arguments:

This method has no arguments.

Return value:

This method does not return anything.

Details:

This method is private. It is meant to be over-ridden by the programmer in her subclass to change the default behavior of turning automatic data field detection on.

Example:

The following example overrides *initAutoDetection* method to switch off auto detection:

```
::class 'AddressBookDialog' subclass ResDialog
::method initAutoDetection
   self~noAutoDetection
```

3.14.16. noAutoDetection

```
>>--noAutoDetection------------------------------><
```

When the *noAutoDetection* method is invoked, it switches off *automatic* data detection.

Arguments:

This method has no arguments.

Return value:

This method does not return anything.

3.14.17. setCheckBoxData

```
>>--setCheckBoxData(--id--,--data--)-------------><
```

The *setCheckBoxData* method sets the data *attribute* of a check box control.

Arguments:

The arguments are:

id [required]

The resource ID of the check box. May be numeric or symbolic.

data [required]

The *data* to be assigned to the underlying check box. The *data* of a check *CheckBox* is defined to be its check state. For normal check boxes this is 0 for not checked and 1 for checked. Three-state check boxes have the additional indeterminate state. The value for the indeterminate state is 2.

Return value:

The return values are:

-1

A symbolic ID was used and it could not be resolved.

0

No error.

1

An error occurred.

Remarks:

This method will work for any check box control, the underlying control does not *have* to be connected to a data attribute. However, the *id* argument must be the resource ID of a check box control. If it is not, say it is the resource ID of a list view control, the results are unpredictable.

Details:

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

3.14.18. setComboBoxData

```
>>--setComboBoxData(--id--,--data--)-------------><
```

The *setComboBoxData* method sets the data *attribute* of a combo box control.

Arguments:

The arguments are:

id [required]

The resource ID of the combo box control. May be numeric or symbolic.

data [required]

The *data* to be assigned to the underlying combo box control. The *data* of a combo *ComboBox* is defined to be the text in the selection field or the selected combo box item.

Return value:

The return values are:

-1

A symbolic ID was used and it could not be resolved.

0

No error.

1

An error occurred.

Remarks:

This method will work for any combo box control, the underlying control does not *have* to be connected to a data attribute. However, the *id* argument must be the resource ID of a combo box control. If it is not, say it is the resource ID of an edit control, the results are unpredictable.

Details:

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

3.14.19. setControlData

```
>>--setControlData(--id--,--dataValue--)---------><
```

The *setControlData* method sets the data *attribute* of a dialog control. The programmer does not have to know what kind of control it is.

Arguments:

The arguments are:

id [required]

The resource ID of the dialog control. May be numeric or symbolic.

dataValue [required]

The data that is assigned to the control. The format **should** match the type of the control.

Return value:

The return values are:

-1

A symbolic ID was used and it could not be resolved.

0

No error.

1

An error occurred.

Remarks:

The dialog control **should** have been previously connected to a data attribute. If the control has not been previously connected to a data attribute, the implementation of this method assumes the control is an edit control and sets the text of the control to *dataValue*. Since all windows can have their text set, this may be okay. However the outcome is dependent on the type of control. Setting the text of a check box will change its label, setting the text of a tree view control will do nothing. The method is intended to only be used with controls connected to a data attribute.

Likewise, if the format of *dataValue* does not match the type of control, the results are unpredictable. It appears that the original intent of this method was to be used for controls such as tree views, list views, tabs, etc.. When these controls were added to ooDialog, no corresponding setTreeViewData(), setTabData(), etc., methods were implemented. The generic *setControlData*() method works for all connected controls, but it works best if the programmer knows the type of control.

A better method to use, if the programmer is insistent on thinking in terms of data attributes, is probably to use the *data =* method of the dialog *control* object.

Details:

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

Example:

The following example sets dialog control with resource ID of 123 to the (string) data "1 2 3". This is meaningful if the control is an edit control, or if it is a list box that contains the line "1 2 3". However, for other controls it is probably meaningless.

```
MyDialog~setControlData(123, "1 2 3")
```

3.14.20. setData

```
>>--setData-------------------------------------><
```

The *setData* method sets the state of all underlying, connected, dialog controls to the values of the matching dialog object data *attribute*s.

Arguments:

This method takes no arguments.

Return value:

0 on success, otherwise 1.

Remarks:

Note that this method sets the state for every connected attribute. This may not always be desirable. For instance the focused node of a tree-view control may be changed. For finely grained control of which dialog controls have their state set, the programmer should use the *setControlData* or *setDataAttribute* methods. Both of these methods transfer the value of a single object attribute that is specified by the programmer.

Details:

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

Example:

Dialog items with ID 102, 201 and 203 are connected to the attributes **`EDIT_1`**, **`DATA201`**, and **`LISTBOX_DAYS`**. Attribute **`DATA201`** is generated by the connectCheckBox method. Then the attributes are initialized with some values. This does not change the state of the controls in the dialog until the *setData* method is invoked.

```
     . .
     .
dialog~connectEdit(102, "EDIT_1")
dialog~connectCheckBox(201,)
dialog~ConnectListbox(203, "LISTBOX_DAYS")
     .
     .
     .
dialog~EDIT_1 = "Memorial Day"
dialog~DATA201 = 1
dialog~LISTBOX_DAYS = "Monday"

dialog~setData
```

3.14.21. setDataAttribute

```
>>--setDataAttribute(--attributeName--)----------><
```

The *setDataAttribute* method transfers the value of a connected data *attribute* of the Rexx dialog object to the underlying dialog control.

Arguments:

The single argument is:

attributeName [required]

The name of the data attribute whose value is transferred to the underlying dialog control.

Return value:

The return values are:

-1
: A symbolic ID was used and it could not be resolved.

0
: No error.

1
: An error occurred.

Remarks:

The terms *transfers* and *receives* the data are holdovers from the original documentation and the original data *attribute* abstraction. What the *setDataAttribute* method actually does is to set the state of the underlying dialog control to the value of the connected data attribute. The defined value for the *data* of any dialog control is documented in the *connnect data attriubte* method for the control.

Details

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

Example:

The following example sets the state the of the connected dialog control to the value of the data attribute DATA101: to

```
self~setDataAttribute("DATA101")
```

3.14.22. setDataStem

```
>>--setDataStem(--dataStem.--)-------------------><
```

Sets the *data*, (the state,) of a number of Windows dialog controls to the values specified by the stem. Each index of *dataStem.* is the resource ID of a dialog control. The control with that ID, within the dialog, has its state set to match the value of the stem variable at that index. The resource ID can be specified using either numeric resource IDs or symbolic resource IDs.

This method only works for controls that have been connected to a *data* attribute. Either by *automatic data detections [156]*, or through one of the *connnect data attriubte*) methods. Every connected control has its state set when this method is invoked. If the data stem, *data.* has no matching index for a connected control, that control's state is set to the empty string. Any indexes in the data stem that do not match the resource ID of a registered control are ignored.

Arguments:

dataStem. [required]
: A stem variable containing the values used to set a dialog control's state. Remember the trailing period.

Details

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

Return value:

This method does not return a value.

Details:

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

Example:

The following example initializes the dialog controls with IDs 21, 22, and 23, provided they have been connected previously:

```
      .
      .
      .
dlgStem.21="Windows 95"
dlgStem.22="0"
dlgStem.23="1 2 3"
self~setDataStem(dlgStem.)
```

3.14.23. setEditData

```
>>--setEditData(--id--,--data--)-----------------><
```

The *setEditData* method sets the data *attribute* of an edit control.

Arguments:

The arguments are:

id [required]

The resource ID of the edit control. May be numeric or symbolic.

data [required]

The *data* to be assigned to the underlying edit control. The *data* of an edit control is defined to be the text entered in the control. If there is no text, the data is the empty string.

Return value:

The return values are:

-1

A symbolic ID was used and it could not be resolved.

0

No error.

1

An error occurred.

Remarks:

This method will work for any edit control, the underlying control does not *have* to be connected to a data attribute. However, the *id* argument must be the resource ID of an edit control. If it is not, say it is the resource ID of a radio button control, the results are unpredictable.

Details:

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

Example:

Assume that three methods are connected to push buttons. The *setToDefault* method sets the text in the underlying edit control with resource ID of 234 to the value *256*. But, it does not change the edit control's connected attribute. Using *setEditData* has the same effect as the user typing text into the edit control. The connecte attribute in the Rexx dialog object (**`DATA234`**) still has the original value. Thus it is possible to undo the changes, (using the *unDoChanges* method,) or to accept the changes, (using the *acceptValues* method.)

```
::method setToDefault
   self~setEditData(234, "256")

::method acceptValues
   self~getDataAttribute(DATA234)

::method undoChanges
   self~setDataAttribute(DATA234)
```

3.14.24. setListBoxData

```
>>--setListBoxData(--id--,--data--)--------------><
```

The *setListBoxData* method sets the data *attribute* of the underly list box to the value specified.

Arguments:

The arguments are:

id [required]

The resource ID of the list box. May be numeric or symbolic.

data [required]

The data that is assigned to the list box. The *data* of a **single-selection** list *ListBox* is defined to be text of the selected item in the list box.

The *data* of a **multi-selection** *ListBox* is defined to be a string containing the item numbers of its selected items, separated by a blank. For a multi-selection list box the string could contain any number of list items depending on what the programmer is setting to be selected.

Return value:

The return values are:

-1

A symbolic ID was used and it could not be resolved.

0

No error.

1

An error occurred.

Remarks:

This method will work for any list box control, the underlying control does not *have* to be connected to a data attribute. However, the *id* argument must be the resource ID of a list box control. If it is not, say it is the resource ID of a list view control, the results are unpredictable.

Details:

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

Example:

The following example is for a single-selection list box. It selects the item whose text is "New York", case insensitive, in the list box with a resource ID of 232:

```
MyBaseDialog~setListBoxData(232, "New York")
```

Example:

The following example selects items 2, 5, and 6 of the multi-selection list box with a resource ID of 345:

```
MyDialog~setListBoxData(345, "2 5 6")
```

3.14.25. setRadioButtonData

```
>>--setRadioButtonData(--id--,--data--)----------><
```

The *setRadioData* method sets the data *attribute* of a radio button control.

Arguments:

The arguments are:

id [required]

The resource ID of the radio button. May be numeric or symbolic.

data [required]

The *data* to be assigned to the underlying radio button. The *data* of a radio *RadioButton* is defined to be the its check state. This is 0 for not checked and 1 for checked.

Return value:

The return values are:

-1

A symbolic ID was used and it could not be resolved.

0

No error.

1

An error occurred.

Remarks:

This method will work for any radio button control, the underlying control does not *have* to be connected to a data attribute. However, the *id* argument must be the resource ID of a radio button control. If it is not, say it is the resource ID of a list view control, the results are unpredictable.

Details:

This method can only be invoked after the underlying Windows dialog has been created. A syntax condition is raised if the underlying dialog does not exist.

3.15. List Box Methods

The following methods deal with list boxes.

3.15.1. addListEntry

```
>>--addListEntry(--id--,--aString--)-------------><
```

The addListEntry method adds a string to the given list box. See also *addComboEntry*. The line is added at the end (by default), or in sorted order if the list box was defined with the sorted flag.

Arguments:

The arguments are:

id

The ID of a list box.

aString

The data to be inserted as a new line.

3.15.2. changeListEntry

```
>>--changeListEntry(--id--,--+-------+--,--aString--)--------------------------><
                             +-index-+
```

The changeListEntry method changes the contents of a line in a list box.

Arguments:

The arguments are:

id

The ID of the list box.

index

The index of the item that you want to replace. If this argument is omitted, the currently selected item is changed.

aString

The new text of the item.

3.15.3. deleteListEntry

```
>>--deleteListEntry(--id--,--index--)------------><
```

The deleteListEntry method deletes an item from a list box. See also *deleteComboEntry*.

Arguments:

The arguments are:

id

The ID of the list box.

index

The line number of the item to be deleted. Use *findListEntry* to retrieve the index of an item. If this argument is omitted, the currently selected item is deleted.

3.15.4. findListEntry

```
>>--findListEntry(--id--,--aString--)------------><
```

The findListEntry method returns the index of the given string within the given list box. The first item has index 1, the second has index 2, and so forth. If the list box does not contain the string, 0 is returned.

Arguments:

The arguments are:

id

The ID of the list box.

aString

The item text you are looking for.

Example:

The following example shows a method that adds the contents of an entry line (214) to the list box (215) if no item with the same value is already contained in it:

```
      .
      .
      .
::method PutEntryInList
   str = self~getEditData(214)
   if self~findListEntry(215, str) = 0 then
      self~addListEntry(215, str)
```

3.15.5. getCurrentListIndex

```
>>--getCurrentListIndex(--id--)------------------><
```

The getCurrentListIndex method returns the index of the currently selected list box item, or 0 if no item is selected. To retrieve the text of the selected list box item use the *getListBoxData*() method.

Arguments:

The only argument is:

id

The ID of the list box.

3.15.6. getListEntry

```
>>--getListEntry(--id--,--index--)---------------><
```

The getListEntry method returns the string at index of the list.

Arguments:

The arguments are:

id

The ID of the list box.

index
: The index of the list entry to be retrieved.

Example:

```
if dlg~getListEntry(203,5)="JOHN"
then ...
```

3.15.7. getListItemHeight

```
DialogExtensions::getListItemHeight


>>--getListItemHeight(--id--)--------------------><
```

3.15.8. getListItemHeightPx

```
DialogExtensions::getListItemHeightPx


>>--getListItemHeightPx(--id--)------------------><
```

3.15.9. getListItems

```
>>--getListItems(--id--)-------------------------><
```

The getListItems method returns the number of items in the list box.

Arguments:

The only argument is:

id
: The ID of the list box.

3.15.10. getListWidth

```
DialogExtensions::getListWidth


>>--getListWidth(--id--)-------------------------><
```

3.15.11. getListWidthPx

```
DialogExtensions::getListWidthPx


>>--getListWidthPx(--id--)-----------------------><
```

3.15.12. insertListEntry

```
>>--insertListEntry(--id--,--+-------+--,--aString--)--------------------------><
                             +-index-+
```

The insertListEntry method inserts a string into the given list box. See also *insertComboEntry*.

Arguments:

The arguments are:

id

The ID of the list box.

index

The index (line number starting with 1) of the item after which the new item is inserted. If this argument is omitted, the new item is inserted after the currently selected item.

aString

The text string to be inserted.

3.15.13. listAddDirectory

```
>>--listAddDirectory(--id--,--drvPath--,--fileAtrs--)-----------><
```

The listAddDirectory method adds all or selected file names of a given directory to the list box. See *comboAddDirectory* for more information.

3.15.14. listDrop

```
>>--listDrop(--id--)-----------------------------><
```

The listDrop method removes all items from the list box.

Arguments:

The only argument is:

id

The ID of the list box.

3.15.15. setCurrentListIndex

```
>>--setCurrentListIndex(--id--+----------+--)----><
                              +-,--index-+
```

The setCurrentListIndex selects the item with the given index in the list. If called without an index, all items in the list are deselected. See *setListBoxData* for information on how to select a list box item using a data value.

Arguments:

The arguments are:

id

The ID of the list box.

index
: The index within the list box.

3.15.16. setListColumnWidth

```
DialogExtensions::setListColumnWidth


>>--setListColumnWidth(--id--,--width--)---------><
```

3.15.17. setListColumnWidthPx

```
DialogExtensions::setListColumnWidthPx


>>--setListColumnWidthPx(--id--,--width--)-------><
```

3.15.18. setListItemHeight

```
DialogExtensions::setListItemHeight


>>--setListItemHeight(--id--,--height--)---------><
```

3.15.19. setListItemHeightPx

```
DialogExtensions::setListItemHeightPx


>>--setListItemHeightPx(--id--,--height--)-------><
```

3.15.20. setListWidth

```
DialogExtensions::setListWidth


>>--setListWidth(--id--,--scrollwidth--)---------><
```

3.15.21. setListWidthPx

```
DialogExtensions::setListWidthPx


>>--setListWidthPx(--id--,--scrollwidth--)-------><
```

3.15.22. setListTabulators

```
                                +-,---+
                               V      |
>>--setListTabulators(--id--,----tab-+--)--------><
```

The setListTabulators method sets the tabulators for a list box. Thus you can use items containing tab characters ("09"x), which is useful for formatting the list in more than one column.

Arguments:

The arguments are:

id

The ID of the list box.

tab

The positions of the tabs relative to the left edge of the list box.

Example:

The following example creates a four-column list and adds a tab-formatted row to the list. The tabulator positions are 10, 20, and 30.

```
MyDialog~setListTabulators(102, 10, 20, 30)
MyDialog~addListEntry(102, var1  "09"x
var2  "09"x  ,
                           var3  "09"x  var4)
```

3.16. Menu Methods

Prior to ooDialog level 4.2.0, menus were really only minimally supported, through a few methods of the dialog class. In ooDialog 4.2.0 true *Menu* objects were introduced. Therefore the menu methods of the dialog object, other than *getMenuBar*() and *hasMenuBar*(), are deprecated. Use the methods of the menu classes directly to work with menus. The listing of the *deprecated* dialog methods documents all the methods to use to replace all deprecated methods.

3.16.1. getMenuBar

```
>>--getMenuBar-----------------------------------><
```

The *getMenuBar* method returns the *Menu* object attached to the dialog, or **.nil** if there is no menu attached.

Arguments:

There are no arguments.

Return value:

If a menu is attached to the dialog, the Rexx menu object is returned. If no menu is attached, then **.nil** is returned.

Example:

This snippet of code comes from a program where the user can customize the interface to use a menu or to not use a menu. When writes should be disabled, the method checks for a menu and if present disables the appropriate menu items.

```
::method onDisableWrites
    if self~hasMenuBar then do
        menu = self~getMenuBar
        menu~disable(IDM_WRITE_TO_FILE)
        menu~disable(IDM_WRITE_TO_MEMORY)
        menu~disable(IDM_WRITE_TO_SOCKET)
    end
    return 0
```

3.16.2. hasMenuBar

```
>>--hasMenuBar-----------------------------------><
```

Tests if the dialog has a menu bar attached.

Arguments:

There are no arguments.

Return value:

True if a menu is attached to the dialog, otherwise false.

Example:

See the example for the *getMenuBar*() method.

3.17. Operating System Related Methods

The methods in this section are used to access operating system functionality. Some of them return information known to, or used by, the operating system when it works with dialogs or dialog controls. Such as returning the window *handle* of a dialog or a dialog control. Other of the methods are used to invoke some functionality provided by the operating system, such as capturing the mouse, or converting dialog units to pixels.

3.17.1. dlgUnit2pixel

```
>>--dlgUnit2pixel(--du--)------------------------><
```

Takes a dimension expressed in dialog units of this dialog and transforms it to a dimension expressed in pixels.

Arguments:

du [required] [In/Out]

The object to transform, can be either a *Point*, *Size*, or *Rect* On input, the unit of measurement is assumed to be dialog units and on return the dialog units will have been converted to pixels

Return value:

Returns true on success and false on error.

Remarks:

This method accurately converts dialog units of this dialog to pixels. Do not use *factorX* and *factorY* for the conversion as they are *inaccurate*.

Normally, this method is invoked prior to the creation of the underlying dialog. In order to be accurate, the *fontName* and *fontSize* attributes must reflect the actual font the dialog will use. Use the *setDlgFont*() method if necessary.

Details:

Raises syntax errors when incorrect arguments are detected.

3.17.2. getSelf

```
>>--getSelf--------------------------------------><
```

The *getSelf* method returns the *handle* of the Windows dialog associated with the Rexx dialog instance.

Arguments:

This method has no arguments.

Return value:

Returns the window handle of the *underlying* Windows dialog for this Rexx dialog, or 0 if the underlying dialog has not been created yet.

Remarks:

Prior to ooRexx 3.2.0, the *getSelf* method was not documented. The *get* method was documented as if it were the *getSelf* method. The *get* method returns the handle to the most recently created dialog. In an application with more than one dialog this may not be the same as the Windows dialog associated with the instance object the programmer is working with.

Example:

Below is an example demonstrating these Windows handles and how they are related. Assume *showHandles* is a method of a subclass of a *RcDialog* and that *staticDlg* is a saved reference to another ooDialog object that was just created. The *showHandles* method is connected to a push button and then displays the handles when the user clicks that button.

```
::method showHandles
  expose staticDlg

  hwndTop = self~get
  hwndMyDialogHandle = self~dlgHandle
  hwndMySelf = self~getSelf
  hwndStatic = staticDlg~dlgHandle
  say "Top dialog:      " hwndTop
  say "Self (dlgHandle):" hwndMyDialogHandle
  say "Self (getSelf):  " hwndMySelf
  say "Static dialog:   " hwndStatic
```

Output:

The above might write to the console something like the following:

```
  Top dialog:       787096
  Self (dlgHandle): 4522228
  Self (getSelf):   4522228
  Static dialog:    787096
```

3.17.3. get

```
>>--get------------------------------------------><
```

The *get* method returns the *handle* of the Windows dialog associated with the top Rexx dialog instance.

Arguments:

This method has no arguments.

Return value:

Returns the window handle of the *underlying* Windows dialog for the top Rexx dialog, or 0 if there is no top dialog.

Remarks:

If more than one ooDialog exists, the top dialog and the dialog whose method is executing may not be the same. When more than one dialog exists, the top dialog is the one that was created last.

Example:

Below is an example demonstrating that the top dialog and the executing dialog may not be the same. Assume *display* is a method of a subclass of a *UserDialog* and that *staticDlg* is a saved reference to another ooDialog object.

```
::method display
  expose staticDlg

  hwndTop = self~get
  hwndSelf = self~dlgHandle
  hwndStatic = staticDlg~dlgHandle
  say "Top dialog:    " hwndTop
  say "Self:          " hwndSelf
  say "Static dialog:" hwndStatic
```

Output:

The above might write to the console something like the following:

```
  Top dialog:    787096
  Self:          4522228
  Static dialog: 787096
```

3.17.4. getControlHandle

```
>>--getControlHandle(--id--+---------+--)--------><
                           +-,--hDlg-+
```

Retrieves the window *handle* of the specified dialog control.

Arguments:

The arguments are:

id [required]

The resource id of the dialog control whose window handle is to be retrieved. May be numeric or *symbolic*.

hDlg [optional]

The window handle of the dialog the control belongs to. By default, it is assumed that the dialog control belongs to this dialog.

Return value:

On success, the return is the window handle of the dialog control with the specified resource ID. On error, 0 is returned.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example get the window handle of the Ok button.

```
hwndOkButton = self~getControlHandle(OK)
```

3.17.5. getControlID

```
>>--getControlID(--hwnd--)-----------------------><
```

Given a valid window *handle* to a dialog control, the *getControlID* method returns the numeric resource ID of the control.

Arguments:

The only argument is:

hwnd [required]

The window handle of the dialog control.

Return value:

Negative values indicate an error.

-1

The *hwnd* argument is not a valid window handle.

less than -1

The value is the negated Operating System Error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

other

The resource ID of the dialog control.

Example:

The following is a complete working example. It allows the user to check a check box with the F2 key and uncheck it with the F3 key. The F2 and F3 key press events are connected to the **check** and **uncheck** methods. When the user presses the F2 key, the program determines which control has the focus, uses the returned window handle to get the control ID, and then uses the ID to check the check box. And the same approach if the F3 key is pressed.

Note that in this dialog, there are only four controls that can have the focus. If the OK button has the focus, then nothing is done. In a more complex application, the programmer would probably check that the resource ID matches one of the check boxes instead.

```

/* Simple Grocery List */
```

```
  dlg = .SimpleDialog~new
  dlg~constDir[IDC_GB_LIST] = 101
  dlg~constDir[IDC_CB_MILK] = 102
  dlg~constDir[IDC_CB_BREAD] = 103
  dlg~constDir[IDC_CB_FRUIT] = 104
  dlg~constDir[IDC_CB_CEREAL] = 105

  if dlg~initCode = 0 then do
    dlg~create(30, 30, 150, 150, "The Simple Grocery List", "VISIBLE")
    dlg~execute("SHOWTOP")
  end

-- End of entry point.
::requires "ooDialog.cls"

::class 'SimpleDialog' subclass UserDialog

::method check

  hwnd = self~getFocus
  id = self~getControlID(hwnd)
  if id == self~constDir[IDOK] then return

  self~newCheckBox(id)~check

::method unCheck

  id = self~getControlID(self~getFocus)
  if id == self~constDir[IDOK] then return

  self~newCheckBox(id)~uncheck

::method defineDialog

  self~createGroupbox(IDC_GB_LIST, 10, 20, 130, 90, , "Check Needed Groceries");
  self~createCheckBox(IDC_CB_MILK, 30, 35, , , "GROUP", "Milk");
  self~createCheckBox(IDC_CB_BREAD, 30, 55, , , "NOTAB", "Bread");
  self~createCheckBox(IDC_CB_FRUIT, 30, 75, , , "NOTAB", "Fruit");
  self~createCheckBox(IDC_CB_CEREAL, 30, 95, , , "NOTAB", "Cereal");

  self~createPushButton(IDOK, 105, 120, 35, 15, "GROUP", "OK")

::method initDialog

  -- We use the 'none' filter so that only a F2 or F3 key press is
  -- captured.  (Not Alt-F2, or Shift-F2, etc..)
  self~connectKeyPress(check, .VK~F2, "NONE")
  self~connectKeyPress(unCheck, .VK~F3, "NONE")
```

3.17.6. getTextSizeDu

```
>>--getTextSizeDu(--text--)----------------------><
```

Calculates the size the given string will occupy in the dialog. The size is returned in *dialog unit*. To calculate the size in pixels use the *getTextSizePx*() method. The *getTextSizeDlg*() method is similar to *getTextSizeDu*, but should only be used when necessary

The primary use of this method would be in laying out the dialog template in a *UserDialog* before the underlying dialog is created. To be sure the calculation is correct, ensure that the dialog font *fontName* are set to the font that will be used by the dialog when it is created. The attributes can be set directly or by using the *setDlgFont*() method.

```
dlg~setDlgFont("fontName", fontSize)
dlg~getTextSizeDu("some text")
```

Arguments:

The only argument is:

text

Calculate the size for this string.

Return value:

The size needed for the string is returned as a *Size* object.

Example:

This example calculates the size needed to display a static message. It then uses that size to initialize a .Rect object with the coordinates of where to place the static control in the dialog. Note that a .Rect object normally defines a bounding rectangle. But, in this case the rectangle is used to specify the upper left corner and the width and the height of the rectangle.

```
::method calcSizes private
  expose msgRect okRect

  -- Get the size needed for the message, place the static
  -- control at (10, 10) in the dialog.

  size = self~getTextSizeDu(message)
  msgRect = .Rect~new(10, 10, size~width, size~height)
  ...

  -- Get a rectangle for the ok push button, we know we
  -- want the width and height to be 40 and 14.

  okRect = .Rect~new(0, 0, 40, 14)

  -- Now calculate its position in relation to the message.
  -- This aligns the right edge of the button with the right
  -- edge of the message and places it 5 dialog units below
  -- the button.

  okRect~left = msgRect~left + msgRect~right  - 40
  okRect~top  = msgRect~top  + msgRect~bottom + 5
  ...

::method defineDialog
  expose message msgRect okRect

  -- Create our dialog controls.
  r = msgRect
  self~createStaticText(IDC_STATIC, r~left, r~top, r~right, r~bottom, , message)

  r = okRect
  self~createPushButton(IDOK, r~left, r~top, r~right, r~bottom, "DEFAULT", "Ok")
```

3.17.7. getTextSizeDlg

```
>>--getTextSizeDlg(--text--+-------------+--+-------------+--+---------+--)---->< 
                           +-,-fontname--+  +-,-fontSize--+  +-,-hwnd--+
```

Calculates the size, (width and height,) in *dialog unit* for a given string.

Note: The convoluted arguments to this method are needed to retain backwards compatibility with older versions of ooDialog. The recommendation is to not use this method, if at all possible. The *getTextSizeDu*() method should be used in all cases where it is feasible.

In general, dialog units are only of value in laying out the dialog controls before the underlying dialog is created. Once the underlying dialog is created, it makes more sense to work with pixels. Since dialog units are tied directly to the font actually used by a specific dialog, specifying a font different from that used by the dialog will usually give incorrect results. Likewise, specifying a window handle other than the window handle of the dialog will also give incorrect results. Because of this, the optional arguments are usually not of much use, making *getTextSizeDu* is easier to use.

One exception to this, is when the text size is being calculated for a dialog control that the programmer intends to set to a font different than the dialog font. In this case, specify the name and size of the font that the dialog control will use.

If this method is used to aid in the lay out of the dialog controls, to be sure the calculation is correct, the dialog font *fontName* must be set to the font that will be used by the dialog when it is created. The attributes can be set directly or by using the *setDlgFont*() method.

```


dlg~setDlgFont("fontName", fontSize)
size = dlg~getTextSizeDlg("some text")
```

The ooDialog programmer is **strongly** urged to use the *getTextSizeDu*() rather than this method.

Arguments:

The arguments are:

text [required]

The string whose size is desired. If none of the optional arguments are specified then the dialog font is used to calculate the size.

fontName [optional]

The name of the font to use to calculate the size needed for the string. Use This argument when the string will be displayed in a dialog control that is using a font **different** than the dialog font. When this argument is used, then the specified font is always used in the calculation.

fontSize [optional]

The size of the font named by the fontName argument. This argument is always ignored when the fontName argument is omitted. If the fontName is specified and this argument is omitted then the default font size is used. (Currently the default size is 8.)

hwnd [optional]

A valid window handle. The font of this window is used to calculate the text size. This argument is always ignored when the fontName argument is specified. As per the notes above the Rexx programmer is encouraged not to use this argument.

Return value:

The size needed for the string is returned in a *Size* object.

3.17.8. getTextSizeTitleBar

```
>>--getTextSizeTitleBar(--text--)----------------><
```

Gets the size, width and height, of a string used in the caption bar for the title.

Arguments:

The single argument is:

text [required]

The string to get the size of. This can be any string, it does not have to be the actual string used for the title. This method returns the size for the string, if it were to be used in the caption bar.

Return value:

Returns a *Size* object containing the width and height of a rectangle that would contain the string.

Remarks:

This method can be used before the underlying Windows dialog is created. The returned size is identical whether the dialog has been created by the operating system or not.

Example:

This example calculates the width, in dialog units, that is required for a dialog such that the title fits without the operating system clipping it:

```
::method getMinDlgCX private
    use strict arg title

    s = self~getTextSizeTitleBar(title)

    padding = 18
    s~width += (2 * .SM~cxFixedFrame) + .SM~cxSize + .SM~cxSmIcon + padding
    self~pixel2dlgUnit(s)

    return s~width
```

3.17.9. pixel2dlgUnit

```
>>--pixel2dlgUnit(--pixels--)--------------------><
```

Takes a dimension expressed in pixels and transforms it to a dimension expressed in dialog units of this dialog.

Arguments:

The single argument is:

pixels [required] [In/Out]

The object to transform, can be either a *Point*, *Size*, or *Rect* On input, the unit of measurement is assumed to be pixels and on return the pixels will have been converted to dialog units.

Return value:

True on succes, false on error.

Remarks:

This method accurately converts pixels to the correct dialog units of this dialog. Do not use *factorX* and *factorY* as they are *inaccurate*.

However, if this method is invoked prior to the creation of the underlying dialog, the *fontName* and *fontSize* attributes must reflect the actual font the dialog will use. Use the *setDlgFont*() method if necessary.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

This example shows how to position controls in a *UserDialog* relative to a static image control. The pixel size of the picture is known. That size is then converted to dialog units so the other controls can be positioned correctly.

```
::method defineDialog

  -- Be sure the font name and size attributes are correct.  We are using MS
  -- Shell Dlg 2 with a point size of 8 to create the dialog:
  self~setDlgFont("MS Shell Dlg 2", 8)

  caption = "Conrad, King of the Hill"

  -- The values used for the size and width of the image are ignored by the OS
  -- when the REALSIZEIMAGE style is used.  The OS will addjust the size of the
  -- static control to match the actual size of the bitmap.
  self~createStaticImage(IDC_BMP_PICTURE, 10, 10, 20, 17, "BITMAP REALSIZEIMAGE")

  -- We know the real size of the bitmap in pixels:
  size = .Size~new(265, 282)

  -- Convert the pixels to dialog units so we can position the other controls.
  self~pixel2dlgUnit(size)

  -- Position the caption, 5 dialog units below the picture, aligned with the
  -- left edge of the picture.
  yPos = 10 + size~height + 5
  self~createStaticText(IDC_ST_DESCRIPTION, 10, yPos, 176, 20, "TEXT LEFT", caption)

  -- Position the Ok button top aligned with the caption and align with the
  -- right edge of the picture.
  xPos = 10 + size~width - 50
  self~createPushButton(IDOK, xPos, yPos, 50, 14, "DEFAULT", "Ok", ok)
```

3.18. Scroll Bar Methods

The methods in this section are dialog object methods used to set or get the behavior of a scroll bar.

3.18.1. determineSBPosition

```
DialogExtensions::determineSBPosition


>>--determineSBPosition(--id--,--posdata--+----------+--+--------+--)----------><
                                          +-,-single-+  +-,-page-+
```

3.18.2. getSBPos

```
DialogExtensions::getSBPos


>>--getSBPos(--id--)-----------------------------><
```

3.18.3. getSBRange

```
DialogExtensions::getSBRange


>>--getSBRange(--id--)---------------------------><
```

3.18.4. setSBPos

```
DialogExtensions::setSBPos


>>--setSBPos(--id--,--pos--+----------+--)-------><
                           +-,-redraw-+
```

3.18.5. setSBRange

```
DialogExtensions::setSBRange


>>--setSBRange(--id--,--min--,--max--,--+--------+--)-------------------------><
                                        +-redraw-+
```

3.19. Sending Window Messages

3.19.1. sendMessageToControl

```
>>--sendMessageToControl(--id--,--msg--,--wParam--,--lParam--)-----------------><
```

The *sendMessageToControl* method sends a Windows message to a dialog control and returns the response as a whole number.

Arguments:

The arguments are:

id [required]

The resource ID of the dialog control. May be numeric or *symbolic*.

msg [required]

The Windows message ID. This may be numeric or a string in conventional *hexadecimal* format.

wParam [required]

The WPARAM message parameter.

lParam [required]

The LPARAM message parameter.

Return value:

The underlying dialog control's response to the message, as a whole number.

Remarks:

This method returns the response as a whole number. This will not be usable if the response is a window *handle*. For those cases, use the *sendMessageToControlH*() method.

Details:

Sets the *.systemErrorCode*.

This method requires an understanding of Windows *message*s.

Example:

The following example checks the radio button with resource ID 9001.

```
MyDialog~sendMessageToControl(9001, "0x000000F1", 1, 0)
```

3.19.2. sendMessageToControlH

```
>>--sendMessageToControlH(--id--,--msg--,--wParam--,--lParam--)----------------><
```

The *sendMessageToControlH* method sends a Windows message to a dialog control and returns the response as a window handle.

Arguments:

The arguments are:

id [required]

The resource ID of the dialog control. May be numeric or *symbolic*.

msg [required]

The Windows message ID. This may be numeric or a string in conventional *hexadecimal* format.

wParam [required]

The WPARAM message parameter.

lParam [required]

The LPARAM message parameter.

Return value:

The underlying dialog control's response to the message, as a window handle.

Remarks:

This method returns the response as a window *handle*. This will not be of much use if the dialog control's response is a whole number. For those cases, use the *sendMessageToControl*() method.

Details:

Sets the *.systemErrorCode*.

This method requires an understanding of Windows *message*s.

Example:

The following example changes the font of the list-view control with symbolic ID of IDC_LV_ADDRESSES and returns the old font. Note that this is exactly what the *getFont*() and *setFont* methods do. In most cases, there is really no need for the Rexx programmer to use the *sendMessageToControl* or *sendMessageToControlH* methods.

```
::method changeFont private
  use strict arg newFont

  oldFont = self~sendMessageToControlH(IDC_LV_ADDRESSES, "0x0031", 0, 0)
  self~sendMessageToControl(IDC_LV_ADDRESSES, "0x0030", newFont, 1)

  return oldFont
```

3.19.3. sendMessageToWindow

```
>>--sendMessageToWindow(--hwnd--,--msg--,--wParam--,--lParam--)----------------><
```

The *sendMessageToWindow* method sends a Windows message to a window and returns the response as a whole number.

Arguments:

The arguments are:

id [required]

The resource ID of the dialog control. May be numeric or *symbolic*.

msg [required]

The Windows message ID. This may be numeric or a string in conventional *hexadecimal* format.

wParam [required]

The WPARAM message parameter.

lParam [required]

The LPARAM message parameter.

Return value:

The underlying window's response to the message, as a whole number.

Remarks:

This method returns the response as a whole number. This will not be usable if the response is a window *handle*. For those cases, use the *sendMessageToWindowH*() method.

Details:

Sets the *.systemErrorCode*.

This method requires an understanding of Windows *message*s.

Example:

The following example sets a new font for the date time picker's month calendar and returns the font it was using. Note that the same thing can be done by using the *getMonthCalFont* and *setMonthCalFont* methods of the *DateTimePicker* class. In most cases, there is really no need for the Rexx programmer to use the *sendMessageToWindow* or *sendMessageToWindowH* methods.

```
::method changeCalendarFont private
  use strict arg hNewFont

  hwnd = self~newDateTimePicker(IDC_DTP_SCHEDULER)~hwnd

  hOldFont = self~sendMessageToWindowH(hwnd, "0x100A", 0, 0)
  self~sendMessageToWindow(hwnd, "0x1009", hNewFont, 1)
```

```
return hOldFont
```

3.19.4. sendMessageToWindowH

```
>>--sendMessageToWindowH(--hwnd,--msg--,--wParam--,--lParam--)-----------------><
```

The *sendMessageToWindowH* method sends a Windows message to a window and returns the response as a *handle*.

Arguments:

The arguments are:

hwnd [required]

The *handle* of the window the message is sent to.

msg [required]

The Windows message ID. This may be numeric or a string in conventional *hexadecimal* format.

wParam [required]

The WPARAM message parameter.

lParam [required]

The LPARAM message parameter.

Return value:

The underlying window's response to the message, as a handle.

Remarks:

This method returns the response as a handle. This will not be of much use if the window's response is a whole number. For those cases, use the *sendMessageToWindow*() method.

Details:

Sets the *.systemErrorCode*.

This method requires an understanding of Windows *message*s.

Example:

The *sendMessageToWindow* *example* also uses the *sendMessageToWindowH* method.

3.20. Window Draw Methods

The methods listed in this section are all related to drawing, redrawing, and clearing window areas. In Windows, the terms drawing and painting are often used to refer to the process of displaying anything on the screen. This includes displaying text.

All drawing operations make use of a *device context*. The ooDialog framework provides a number of methods, such as the *getWindowDC*, *getDC*, and *getControlDC* methods that allow the programmer to retrieve a device context.

3.20.1. clearControlRect

```
DialogExtensions::clearControlRect


>>--clearControlRect(--id--)---------------------><
```

3.20.2. clearRect

```
DialogExtensions::clearRect


Form 1:

>>--clearRect(--hwnd--,--rectangle--)------------><


Form 2:

>>--clearRect(--hwnd--,--pt1--,--pt2--)----------><


Form 3:

>>--clearRect(--hwnd--,--x-,--y-,--cx-,--cy--)---><


Generic form:

>>--clearRect(--hwnd--,--rectCoordinates--)------><
```

3.20.3. clearWindowRect

```
DialogExtensions::clearWindowRect


>>--clearWindowRect(--hwnd--)--------------------><
```

3.20.4. createBrush

```
DialogExtensions::createBrush


>>--createBrush(--+---------+--+-----------------+--)------------------------><
                  +--color--+  +-,-brushSpecifier-+
```

3.20.5. drawButton

```
DialogExtensions::drawButton


>>--drawButton(--id--)---------------------------><
```

3.20.6. freeControlDC

```
DialogExtensions::freeControlDC


>>--freeControlDC(--id--,--dc--)-----------------><
```

3.20.7. freeWindowDC

```
DialogExtensions::freeWindowDC


>>--freeWindowDC(--hwnd--,--dc--)----------------><
```

3.20.8. getControlDC

```
DialogExtensions::getControlDC


>>--getControlDC(--id--)-------------------------><
```

3.20.9. getWindowDC

```
DialogExtensions::getWindowDC


>>--getWindowDC(--hwnd--)------------------------><
```

3.20.10. redrawControl

```
DialogExtensions::redrawControl


>>--redrawControl(--id--+-------------+--)-------><
                        +-,--erasebkg-+
```

3.20.11. redrawRect

```
DialogExtensions::redrawRect


>>--redrawRect(--+------+-,-left-,-top-,-right-,-bottom-,-+-------+--)---------><
                 +-hwnd-+                                 +-erase-+
```

3.20.12. redrawWindow

```
DialogExtensions::redrawWindow
```

```
>>--redrawWindow(--hwnd--)-----------------------><
```

3.20.13. redrawWindowRect

```
DialogExtensions::redrawWindowRect


>>--redrawWindowRect(--+--------+--+------------+--)---------------------------><
                       +--hwnd--+  +-,-erasebkg-+
```

3.20.14. scrollInControl

```
DialogExtensions::scrollInControl


>>--scrollInControl(--id-,-text--+--------+--+--------+--+--------+--+-----+--->
                                 +-,-font-+  +-,-size-+  +-,-opts-+  +-,-y-+

>--+--------+--+---------+--+---------+---------------------------------------><
   +-,-step-+  +-,-sleep-+  +-,-color-+
```

3.20.15. scrollText

```
DialogExtensions::scrollText


>>--scrollText(--hwnd-,-text--+--------+--+--------+--+--------+--+-----+------>
                              +-,-font-+  +-,-size-+  +-,-opts-+  +-,-y-+

>--+--------+--+---------+--+---------+---------------------------------------><
   +-,-step-+  +-,-sleep-+  +-,-color-+
```

3.20.16. writeToWindow

```
DialogExtensions::writeToWindow


>>--writeToWindow(-h-,-x-,-y-,-txt-+-------+-+------+-+--------+-+------+-+------+-)--><
                                   +-,-fnt-+ +-,-fs-+ +-,-opts-+ +-,-fg-+ +-,-bk-+
```

3.20.17. writeToControl

```
DialogExtensions::writeToControl


>>--writeToControl(-id-,-x-,-y-,-txt-+-------+-+------+-+--------+-+------+-+------+-)-><
                                     +-,-fnt-+ +-,-fs-+ +-,-opts-+ +-,-fg-+ +-,-bk-+
```

Base and Mixin Classes

ooDialog programs primarily deal with dialog and dialog control objects. The top-level objects in the programs, say a user *UserDialog* or a button *Button* are actually composed of a number of base objects and / or mixin classes. All dialog and dialog control objects contain the methods of their component base classes.

To make it easier to understand the methods and attributes that all dialogs have in common, this documentation treats the *dialog* object as a whole. The dialog object chapter shows all the methods and attributes that every dialog has, without making much distinction as to what specific mixin or base class the method belongs to. The dialog *control* object, is treated in a similar fashion in its chapter.

This chapter describes the base and mixin classes that make up the dialog object and the dialog control object and shows how they fit together. In addition, it contains the detailed information for the mixin classes inherited by both the dialog and dialog control object. The *dialog* object and the dialog *control* object chapters link back to this chapter for the detailed information where needed.

4.1. Component Classes Diagram

The diagram shown here shows the relation between the dialog and the dialog control objects and their respective component classes. The abstract classes are shown in red. The mixin classes in green. And the concrete subclasses in blue.

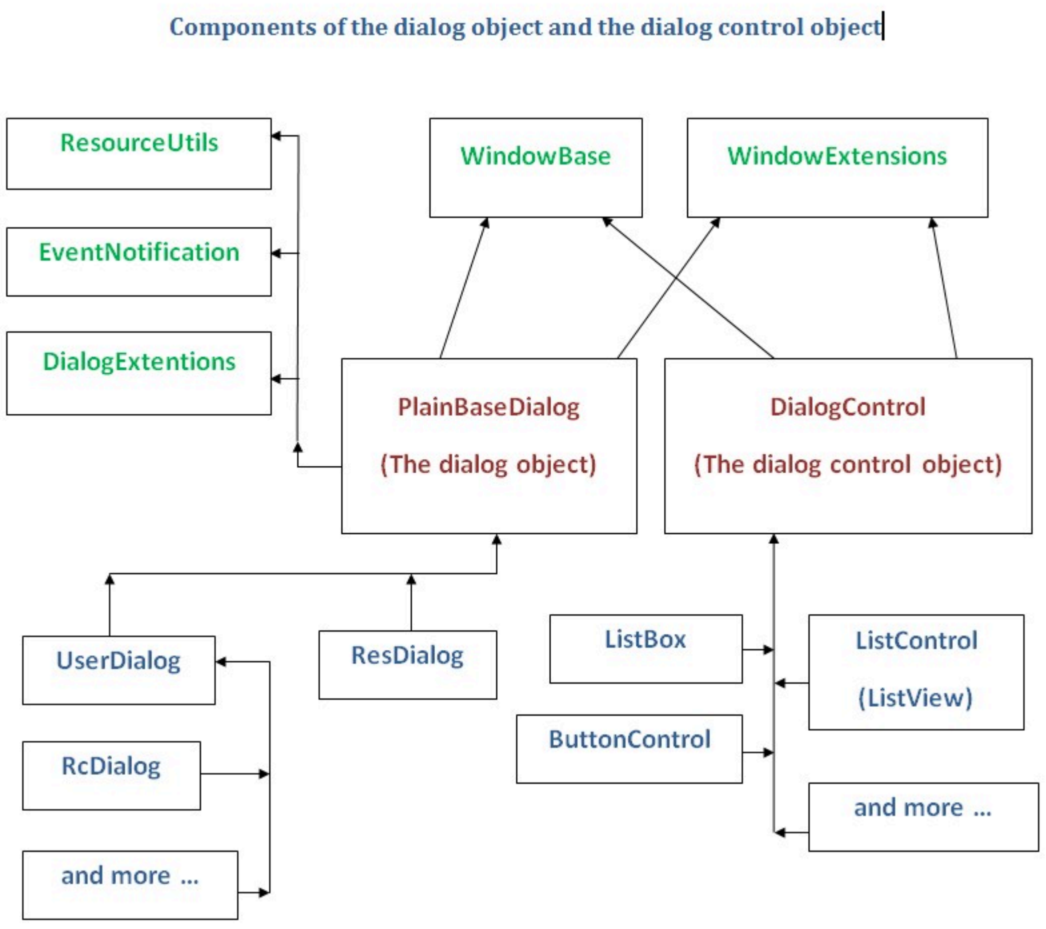


Figure 4.1. Components

4.2. CreateWindows Mixin Class

The *CreateWindows* class is a *mixin* class that has methods to directly create dialog control windows and add them to the *underlying* Windows dialog. The *CreateWindows* class must be inherited by a dialog class to work correctly. If it is inherited by any other class, invoking any of its methods will raise an error condition.

In general in ooDialog, dialog controls are added to a dialog by adding the control to the dialog template. Either through the use of a resource editor for a *RcDialog* and a *ResDialog*, or by programmatically placing them in the template for a *UserDialog*. In earlier versions of ooDialog, the only way to add a dialog control to a dialog was to add it to the dialog template.

Unfortunately, some dialog controls are not generally supported by resource editors. The *StatusBar* and *ToolBar* controls are good examples of this. The controls can be added to a resource script by manually editing the script, which works fine. However, typically C / C++ Windows programmers create these controls directly when they want to use them in their dialogs. The *CreateWindows* class allows the Rexx programmer to directly create dialog controls in a similar fashion.

4.2.1. Method Table

The following table lists the instance methods of the **`CreateWindows`** class:

Table 4.1. CreateWindows Class Method Reference

Method	Description
Class Methods	**Class Methods**
Instance Methods	**Instance Methods**
createReBarWindow	Creates a *ReBar* control in the dialog.
createStatusBarWindow	Creates a *StatusBar* control in the dialog.
createToolBarWindow	Creates a *ToolBar* control in the dialog.

4.2.2. createReBarWindow

```
>>--createReBarWindow(--id--+----------+--)------><
                            +-,-style--+
```

Creates a *ReBar* control in the dialog.

Arguments:

The arguments are:

id [required]

The resource ID of the rebar control. May be numeric or *symbolic*.

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces. Case is not significant:

AUTOSIZE	TOOLTIPS	GROUP
BANDBORDERS	VARHEIGHT	HIDDEN
DBLCLKTOGGLE	VERTICALGRIPPER	TAB

FIXEDORDER	BORDER
REGISTERDROP	DISABLED

AUTOSIZE
: The rebar control will automatically change the layout of the bands when the size or position of the control changes. An AUTOSIZE notification will be sent when this occurs.

BANDBORDERS
: The rebar control displays narrow lines to separate adjacent bands.

DBLCLKTOGGLE
: The rebar band will toggle its maximized or minimized state when the user double-clicks the band. Without this style, the maximized or minimized state is toggled when the user single-clicks on the band.

FIXEDORDER
: The rebar control always displays bands in the same order. You can move bands to different rows, but the band order is static.

REGISTERDROP
: The rebar control generates GETOBJECT notification messages when an object is dragged over a band in the control. Although the ooDialog framework supports this style and event notification, there is no way within the framework to make use of it.

TOOLTIPS
: This style keyword is for completeness, the rebar control itself does not yet support tool tips.

VARHEIGHT
: The rebar control displays bands at the minimum required height, when possible. Without this style, the rebar control displays all bands at the same height, using the height of the tallest visible band to determine the height of other bands.

VERTICALGRIPPER
: The size grip will be displayed vertically instead of horizontally in a vertical rebar control. This style is ignored for rebar controls that do not have the CCS_VERT style.

BORDER
: Adds the *border* window style. By default the BORDER style is not added. However, rebar controls seem to ignore this style completely. The *createRebar* method accepts this keyword so that the ooDialog programmer can experiment with it. But it is not expected that using BORDER will produce any noticeable difference in appearance.

DISABLED
: Gives the rebar control the *disabled* window style. By default the control is enabled

GROUP
: The *group* control style is added to the control. By default the GROUP style is not used.

HIDDEN
: The not *visible* window style.

TAB
: Create the control with the *tabstop* control style. By default the control is created without the TABSTOP style. Note that the behavior of the rebar control does not change with, or without, the TABSTOP style. The *createRebar* method accepts this keyword so that the

ooDialog programmer can experiment with it. But it is not expected that using TAB will produce any noticeable difference in behavior.

In addition, the *createReBar* method has support for any *Common Control Styles* keyword.

If the *style* argument is omitted, the default style is *TABSTOP VARHEIGHT BANDBORDERS CCS_NODIVIDER*.

Return value:

An instantiated Rexx status bar object on success, or the **`.nil`** object on error.

Remarks:

Note that a rebar window ignores any position or size coordinates when it is created. It automatically positions and sizes itself. That is why there are no arguments to this method for the position or size of the status bar.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example is from a **`ResDialog`** program. Since the resource editor used to create the resource only DLL for the program only supported rebars in a very limited way, the programmer elected to create the rebar by using the **`CreateWindows`** mixin class:

```
::class 'RebarDlg' subclass ResDialog inherit CreateWindows

::method initDialog

    ...

  rb = self~createReBarWindow(IDC_RBAR, 'VARHEIGHT BANDBORDERS CCS_NODIVIDER')
```

4.2.3. createStatusBarWindow

```
>>--createStatusBarWindow(--id--+----------+--)--><
                                +-,-style--+
```

Creates a *StatusBar* control in the dialog.

Arguments:

The arguments are:

id [required]

The resource ID of the status bar control. May be numeric or *symbolic*.

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces. Case is not significant:

SIZEGRIP	DISABLED	TAB
TOOLTIPS	GROUP	
BORDER	HIDDEN	

SIZEGRIP
: The status bar control includes a sizing grip at the right end of the status bar. A sizing grip is like a sizing border, it is a rectangular area that the user can click and drag to resize the status bar.

TOOLTIPS
: This style enables ToolTips.

BORDER
: Status bars usually do not have The *border* window style, they draw their own border. The BORDER adds an extra border to the status bar.

DISABLED
: Adds the *disabled* window style.

GROUP
: The status bar will have the *group* control style. By default the status bar does not have the group style.

HIDDEN
: The not *visible* window style.

TAB
: Create the control with the *tabstop* control style. By default the control is created without the tabstop style.

In addition, *createStatusBar* has support for any *Common Control Styles* keyword.

If the *style* argument is omitted, the default style is *SIZEGRIP TOOLTIPS* .

Return value:

An instantiated Rexx status bar object on success, or the **`.nil`** object on error.

Remarks:

Note that a status bar window ignores any position or size coordinates when it is created. It automatically positions and sizes itself. That is why there are no arguments to this method for the position or size of the status bar.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example is from a **`ResDialog`** program. Since the resource editor used to create the resource only DLL for the program did not support status bars, the programmer elected to create the status bar by using the **`CreateWindows`** mixin class:

```
::class 'StatusBarDlg' subclass ResDialog inherit CreateWindows

::method initDialog

    ...

  statusBar = self~createStatusBarWindow(IDC_STATUSBAR, 'BORDER SIZEGRIP')
```

4.2.4. createToolBarWindow

```
>>--createStatusBarWindow(--id--+---------+--)--><
                                +-,-style--+
```

Creates a *StatusBar* control in the dialog.

Arguments:

The arguments are:

id [required]
: The resource ID of the status bar control. May be numeric or *symbolic*.

style [optional]
: A list of 0 or more of the following *style* keywords separated by spaces. Case is not significant:

SIZEGRIP	DISABLED	TAB
TOOLTIPS	GROUP	
BORDER	HIDDEN	

SIZEGRIP
: The status bar control includes a sizing grip at the right end of the status bar. A sizing grip is like a sizing border, it is a rectangular area that the user can click and drag to resize the status bar.

TOOLTIPS
: This style enables ToolTips.

BORDER
: Status bars usually do not have The *border* window style, they draw their own border. The BORDER adds an extra border to the status bar.

DISABLED
: Adds the *disabled* window style.

GROUP
: The status bar will have the *group* control style. By default the status bar does not have the group style.

HIDDEN
: The not *visible* window style.

TAB
: Create the control with the *tabstop* control style. By default the control is created without the tabstop style.

In addition, *createStatusBar* has support for any *Common Control Styles* keyword.

If the *style* argument is omitted, the default style is *SIZEGRIP TOOLTIPS* .

Return value:

An instantiated Rexx status bar object on success, or the `.nil` object on error.

Remarks:

Note that a status bar window ignores any position or size coordinates when it is created. It automatically positions and sizes itself. That is why there are no arguments to this method for the position or size of the status bar.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example is from a **`ResDialog`** program. Since the resource editor used to create the resource only DLL for the program did not support status bars, the programmer elected to create the status bar by using the **`CreateWindows`** mixin class:

```
::class 'RebarDlg' subclass ResDialog inherit CreateWindows

::method initDialog

    ...

  statusBar = self~createStatusBarWindow(IDC_STATUSBAR, 'BORDER SIZEGRIP')
```

4.3. CustomDraw Mixin Class

Custom draw is a service provided by some of the Windows common controls. This service allows the programmer greater control over the appearance of a control than that provided directly though the styles and methods of the control. Not all controls support custom draw, but those that do provide the service in a generic manner through the use of custom draw *event* notifications.

The following Windows controls provide custom draw services:

- Header controls.
- List-view controls.
- Rebar controls.
- Toolbar controls.
- Trackbar controls.
- Tree-view controls.

The current implementation of custom draw in ooDialog supports *simple* custom draw for the *list-view* and *tree-view* controls.

4.3.1. Custom Draw Overview

All windows in the Windows operating system periodically paint and erase themselves based on messages received from the system or other applications. This process of painting or erasing is called a paint cycle. The common controls that support custom draw send *event* notifications at specific points while the control is being drawn. The notifications describe drawing operations that apply to the entire control as well as drawing operations specific to items within the control.

Part of the information sent in the custom draw notification is what stage of the paint cycle the control is in. The stage will either be part of the overall painting of the control or part of the painting of an item the control contains. The stage is identified by a constant value supplied by the **`CustomDraw`** class. The possible draw stages are described in the following table:

Table 4.2. Stages of Custom Draw

Constant Value	Description
Global	**Draw Stages**
CDDS_PREPAINT	This stage is before the paint cycle begins.
CDDS_POSTPAINT	This stage is after the paint cycle is complete.
CDDS_PREERASE	This stage is before the erase cycle begins.
CDDS_POSTERASE	This stage is after the erase cycle is complete.
Item Specific	**Draw Stages**
CDDS_ITEMPREPAINT	This stage is before an item is drawn.
CDDS_ITEMPOSTPAINT	This stage is after an item has been drawn.
CDDS_ITEMPREERASE	This stage is before an item is erased.
CDDS_ITEMPOSTERASE	This stage is after an item has been erased.
CDDS_SUBITEMPREPAINT	This stage is before an subitem is drawn. Only used in *ListView* controls in report mode.
CDDS_SUBITEMPOSTPAINT	This stage is after an subitem has been drawn. Only used in list-view controls in report mode.

The ooDialog program that uses custom draw processes the information sent by the control and returns information that instructs the underlying dialog control what to do. The key to using custom draw is in this information sent back to the control in response to the event notification message. The information returned by the program determines the control's behavior for that paint cycle.

Part of the information returned is a combination of response codes to the event notification. The **`CustomDraw`** class provides constant values for these codes as summarized in the following table:

Table 4.3. Notification Responses

Constant Value	Effect
CDRF_DODEFAULT	The control will draw itself. It will not send additional event notifications for this paint cycle. This response cannot be used with any other response.
CDRF_NOTIFYITEMDRAW	The control will send event notifications for any item-specific drawing operations. It will send event notification messages before and after it draws items.
CDRF_NOTIFYPOSTPAINT	The control will send an event notification when the painting cycle for the entire control is complete.
CDRF_NOTIFYSUBITEMDRAW	The control will send event notifications for any subitem-specific drawing operations. It will send event notification messages before and after it draws subitems.
CDRF_SKIPDEFAULT	The control will not perform any painting at all.
CDRF_DOERASE	The control will only draw the background.
CDRF_SKIPPOSTPAINT	The control will not draw the focus rectangle around an item.

Custom draw can be used by the program to draw the entire item, or to change part of the way an item is drawn. Custom draw can be used to change the foreground and background colors, and the font used by the control for each item. For list-view controls in report view, the programmer can also change change the colors and the font for each individual subitem.

4.3.2. Custom Draw in ooDialog

Using custom draw in ooDialog involves a few basic steps.

Inherit `CustomDraw`

The dialog subclass where the programmer wants to use custom draw must inherit the **`CustomDraw`** mixin class.

```
::class 'CustomDrawDlg' subclass RcDialog inherit CustomDraw
```

The dialog subclass does not have to be a *RcDialog* of course. It can be any dialog class supported by ooDialog, except the *PropertySheetDialog*. Recall that the property sheet dialog is managed by the operating system, not ooDialog. However, the pages of the property sheet dialog, any *PropertySheetPage* dialogs, can use custom draw.

Initialize `CustomDraw`

Before any other use of custom draw, the **`CustomDraw`** class must be initialized. This is done through the *customDraw* method. This can be done wherever it seems convenient, but typically it would be done before the dialog control is populated with its items.

```
::method init
    expose textRed textBlack

    forward class (super) continue

    self~customDraw
```

Register Control(s)

Once the **`CustomDraw`** class is initialized, the next step is to register the dialog controls that will use custom draw. Each individual control must be registered using the *customDrawControl* method.

```
::method init
    ...

    self~customDrawControl(IDC_LV_CUSTOM, 'ListView', onCustomDraw)
```

Code an Event Handler

The last part of using custom draw is coding the event handler. In ooDialog, the information from the Windows dialog control is passed on the Rexx event handler in the form of an object. Each individual dialog control type has a control specific class that is used to pass the information. Details on how to code the event handler are explained in the documentation of the control specific class.

Supported Dialog controls.

ooDialog supports custom draw for the dialog controls listed in the following table. The table has a link to the class used to provide the event handler with the information sent by the underlying dialog control and a link to the documentation for the event handler for the specific type of control.

Table 4.4. ooDialog Supported Custom Draw Controls

Control	Information Class	Event Handler
ListView	*LvCustomDrawSimple*	*List-view CustomDraw Event Handler*
TreeView	*TvCustomDrawSimple*	*TreeView CustomDraw Event Handler*

4.3.3. Simple Custom Draw

Currently, ooDialog supports a limited subset of the full functionality supplied by the Windows custom draw API. The developers are denoting this as *simple* custom draw. Simple custom draw allows the ooDialog programmer to change the foreground and background colors of individual dialog control items and subitems. It also allows the programmer to change the font of individual items and subitems.

There are a number of reasons for only supporting simple custom draw at this point. For one, it is probably sufficient for everything an ooDialog programmer will want to do. It allows the most useful functionality of custom draw to be delivered sooner and leaves the possibility of future enhancements providing more functionality. Drawing in Windows needs to be done quickly and efficiently. This type of programming is normally done in a low-level language. It is not clear that a complete implementation of custom draw would produce satisfactory results.

In simple custom draw, the ooDialog framework does not invoke the custom draw event handler for every single custom draw notification sent by the dialog control. Rather, it replies directly for a number of the drawing stages itself, returning the correct information so that the item and subitem prepaint notifications are sent by the control. It then invokes the Rexx event handler for the item and subitem prepaint notifications. This allows the ooDialog programmer to change the colors and fonts for any individual item or subitem.

In simple custom draw, the Rexx event handler will not receive notifications for the postpaint, preerase, and posterase drawing stages. The ooDialog programmer just needs to focus on handling the prepaint notifications for items and subitems.

4.3.4. Method Table

The following table lists the class and instance methods of the **`CustomDraw`** class:

Table 4.5. CustomDraw Class Method Reference

Method	Description
Constant	**Methods**
constants	The **`CustomDraw`** class provides a number of constant values needed to work with the custom draw interface.
Instance	**Methods**
customDraw	Initializes the custom draw interface in ooDialog.
customDrawControl	Registers the specified dialog control to use custom draw.
rgb	Constructs a COLORREF from the red, green, and blue values of a color..

4.3.5. Constants

The Windows API for custom draw has a number of constant values used to specify varying things. There are constants used to indicate the current stage of custom draw paint cycle, constants to use in replying to custom draw event notifications, etc.. The **`CustomDraw`** class supplies the following *constant* values to use with custom drawing. Note that, except where indicated, the CDRF_* response codes can be combined together. The *or* method can be used to combine values.

CDDS_ITEMPREERASE
: Part of the information sent in a custom draw event notification. It indicates the current draw stage. This is sent before an item is erased.

CDDS_ITEMPREPAINT
: Part of the information sent in a custom draw event notification. It indicates the current draw stage. This is sent before an item is drawn.

CDDS_ITEMPOSTERASE
: Part of the information sent in a custom draw event notification. It indicates the current draw stage. This is sent after an item has been erased.

CDDS_ITEMPOSTPAINT
: Part of the information sent in a custom draw event notification. It indicates the current draw stage. This is sent after an item has been drawn.

CDDS_POSTERASE
: Part of the information sent in a custom draw event notification. It indicates the current draw stage. This is sent when the erase cycle is complete.

CDDS_POSTPAINT
: Part of the information sent in a custom draw event notification. It indicates the current draw stage. This is sent when the paint cycle is complete.

CDDS_PREERASE
: Part of the information sent in a custom draw event notification. It indicates the current draw stage. This is sent before the erase cycle begins.

CDDS_PREPAINT
: Part of the information sent in a custom draw event notification. It indicates the current draw stage. This is sent before the paint cycle begins.

CDDS_SUBITEMPREPAINT
: Part of the information sent in a custom draw event notification. It indicates the current draw stage. This is sent before a subitem is drawn. This only applies to list-view controls in report view.

CDDS_SUBITEMPOSTPAINT
: Part of the information sent in a custom draw event notification. It indicates the current draw stage. This is sent after a subitem has been drawn. It is only sent for list-view controls in report view.

CDRF_DODEFAULT
: One of the response values that can be returned to the dialog control from the custom draw event handler. It indicates that the control should draw itself. The control will not send any more notification messages for the current paint cycle. This response can not be combined with other response values.

CDRF_DOERASE
: One of the response values that can be returned to the dialog control from the custom draw event handler. It indicates that the control should only draw the background.

CDRF_NEWFONT

One of the response values that can be returned to the dialog control from the custom draw event handler. It indicates that the programmer has changed the font for the item being drawn. For list-view and tree-view custom draw, it should also be used to indicate that the colors have been changed.

CDRF_NOTIFYITEMDRAW

One of the response values that can be returned to the dialog control from the custom draw event handler. It indicates the control should send notification message for any item-specific drawing operations. The control will send event notifications before and after it draws items.

CDRF_NOTIFYPOSTPAINT

One of the response values that can be returned to the dialog control from the custom draw event handler. It indicates the control should send a notification when the painting cycle for the entire control is complete.

CDRF_NOTIFYSUBITEMDRAW

One of the response values that can be returned to the dialog control from the custom draw event handler. It indicates the control should send notification message for any subitem-specific drawing operations. The control will send event notifications before and after it draws subitems.

CDRF_SKIPDEFAULT

One of the response values that can be returned to the dialog control from the custom draw 8event handler. It indicates that the control should not do any painting, the application will paint the entire control.

CDRF_SKIPPOSTPAINT

One of the response values that can be returned to the dialog control from the custom draw event handler. It indicates that the control should not draw the focus rectangle around an item.

CLR_DEFAULT

A special COLORREF value. COLORREFs are discussed in the documentation for both the *rgb* and *colorRef* methods. The CLR_DEFAULT constant value is identical to the value returned either of those methods when the CLR_DEFAULT keyword is used as the first argument to the method.

CLR_INVALID

A special COLORREF value. COLORREFs are discussed in the documentation for both the *rgb* and *colorRef* methods. The CLR_INVALID constant value is identical to the value returned either of those methods when the CLR_INVALID keyword is used as the first argument to the method.

CLR_NONE

A special COLORREF value. COLORREFs are discussed in the documentation for both the *rgb* and *colorRef* methods. The CLR_NONE constant value is identical to the value returned either of those methods when the CLR_NONE keyword is used as the first argument to the method.

4.3.6. customDraw

```
>>--customDraw-----------------------------------><
```

Initializes the custom draw interface in ooDialog and allows dialog controls to register for custom draw.

Arguments:

There are no arguments for this method.

Return value:

Returns true on success and false on error. An error is highly unlikely.

Remarks:

The *customDraw* method must be invoked before any other use of custom draw. It performs the basic initialization that allows ooDialog to interact with the Windows custom draw service. The basic sequence is to invoke the *customDraw* method and then register the control(s) that will use custom draw.

Example:

This example initializes the custom draw interface and then registers a list-view control to use custom draw.

```
::method init
    expose textRed textBlack

    forward class (super) continue

    self~customDraw
    self~customDrawControl(IDC_LV_CUSTOM, 'ListView', onCustomDraw)
```

4.3.7. customDrawControl

```
>>--customDrawControl(-id--+-----------+--+------------+-+------------+--)-----><
                           +-,-ctrlName+  +-,-methName-+ +-,-reserved-+
```

Specifies that the programmer will custom draw the named control.

Arguments:

The arguments are:

id [required]

Resource ID of the control, may be numeric or *symbolic*. A syntax condition is raised if the ID is symbolic and it can not be resolved.

ctrlName [optional]

The name of the type of control. I.e., treeView, TrackBar, LISTVIEW. By default the control is assumed to be a ListView. Case is not significant. Only ListView and TreeView controls are currently supported.

methName [optional]

The name of a method in the Rexx dialog that will be invoked for the custom draw event. If omitted it defaults to *onCustomDraw*.

reserved [optional]

Reserved for future use. This argument is currently ignored.

Return value:

Returns 0 on success and 1 on error.

Remarks:

The *customDrawControl* method in effect registers a specific dialog control to receive custom draw event notifications and to use a certain *type* of custom draw. Currently the only type implemented in ooDialog is *simple* custom draw.

It is important that the programmer correctly names the control using custom draw through the *ctrlName* argument. Failure to do so will result in undefined behaviour.

The event handling method for the custom draw event will receive a single object as its argument. The object conveys information to the event handler concerning the event and receives information from the event handle to pass back to the underlying Windows control. The exact object will depend on the type of dialog control. List-views will receive a *LvCustomDrawSimple* object and tree-views will receive a *TvCustomDrawSimple* object.

It is possible to have several dialog controls register to use the same event handling method. The information object sent to the event handler has the resource ID of the control the event is for. However, the event handler needs to respond to the events as quickly as possible. Therefore it may be better to use a different event handler for each dialog control and eliminate the need to decipher which control the event is for in a single handler.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example initializes the custom draw interface and then registers two controls to use custom draw:

```
self~customDraw
self~customDrawControl(IDC_LV_A, 'ListView', onCustomDrawA)
self~customDrawControl(IDC_LV_B, 'ListView', onCustomDrawB)
```

4.3.8. rgb

```
>>--rgb(--+---+--+-----+--+-----+--)-------------><
          +-r-+  +-,-g-+  +-,-b-+
```

Constructs a COLORREF from the red, green, and blue components of a color. In the Windows API values of the type COLORREF are often used for colors.

Arguments:

The arguments are:

r [optional]

The red component for the color. The default is 0.

This argument can also be one of 3 keywords: CLR_DEFAULT, CLR_NONE, or CLR_INVALID, case is not significant. If a keyword is used, the other arguments are ignored. See the remarks section for more information on the keywords.

g [optional]

The green component for the color. The default is 0.

b [optional]

The blue component for the color. The default is 0.

Return value:

Returns the COLORREF value for the specified arguments.

Remarks:

A COLORREF is a 32-bit number with a hexadecimal format of: **`0x00bbggrr`**. The *rgb* method provides a way to construct a COLORREF from the red, green, and blue values. Each color value (red, green, or blue) is a whole number in the range of 0 to 255 inclusive.

The CLR_DEFAULT, CLR_NONE, etc., keywords represent special COLORREF values defined in the Windows API. The exact meaning of the value seems dependent on the context it is used in. For instance, in the list-view custom draw interface, if an item's foreground or background color is assigned CLR_DEFAULT, the list-view draws that item in its default color. On the other hand, if the same thing is done for a tree-view item, the tree-view draws that item in black.

CLR_DEFAULT and CLR_INVALID are actually the same value. The CLR_INVALID keyword is provided for completeness.

The functionality of the *rgb* method is identical to the functionality of the *colorRef* class method of the *Image* class. Either method will return the same value for the same inputs.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example specifies the custom draw colors for a list-view item. The item will be drawn with dark blue text on a light blue background.

```
if list~isChecked(item) then do
    lvcds~clrText   = self~RGB( 17,   5, 250)
    lvcds~clrTextBk = self~RGB(115, 245, 186)
end
```

4.4. DialogControl

In a similar relationship as the *PlainBaseDialog* and dialog objects, the *DialogControl* is the base class of all dialog control classes in the ooDialog framework. It implements methods that are common to every dialog control. DialogControl is also an abstract class. A programmer can not instantiate a new DialogControl object. Rather, one of the concrete dialog control classes such as, the *DateTimePicker* or the *ListView*, are used.

See *Figure 4.1, “Components”* to visualize the mixin classes that the DialogControl inherits. In essence the DialogControl is the dialog *control* object. All the methods and attributes of the DialogControl are listed and documented in the dialog control object chapter. This includes the methods and attributes of its inherited mixin classes.

The DialogControl inherits the following mixin classes:

- *WindowBase*
- *WindowsExtensions*

4.4.1. Method Table

The following table lists the attribute methods and instance methods of the **`DialogControl`** class:

Table 4.6. DialogControl Method Reference

DialogControl Method	Description
Attributes	

DialogControl Method	Description
hDlg	Reflects the window handle of the underlying Windows dialog that the dialog control belongs to.
id	Reflects the numeric resource ID for the dialog control.
oDlg	Reflects the Rexx dialog object that the dialog control belongs to.
Instance Methods	
assignFocus	Sets the input focus to this dialog control.
clearRect	Clears the specified rectangular within the client area of this control.
connectCharEvent	Connects a character event notification sent to the control to a method in the Rexx dialog.
connectFKeyPress	Connects all F Key key press events to a method in the Rexx dialog.
connectKeyPress	Connects a key press event notification with a method in the Rexx dialog.
data	Reprieves the *data* (the value) of the underlying control's state.
Section 4.4.8, “data=”	Sets the state of the underlying dialog control to the *data* (the value) specified.
disconnectKeyPress	Disconnects a method in the Rexx dialog from a previously connected key press event.
getTextSizeDlg	Calculates the size, (width and height,) in dialog units for a given string.
group	Adds or removes the *group* style for the control.
hasKeyPressConnection	Queries if a specific method, or any method, in the Rexx dialog has a connection to a key press event.
redrawRect	Immediately redraws the rectangle of the client area of the associated dialog.
setColor	Sets the background color, and optionally the text color, for this dialog control.
setSysColor	Sets the background color, and optionally the text color, for this dialog control using the system colors.
tabStop	Add or remove the tab stop style for the control.
textSize	Computes the width and height in pixels of the specified string of text when displayed by this control.
useUnicode	Sets the format flag telling the control to use, or not use, Unicode.
usingUnicode	Determines if the control is using Unicode or not.

4.4.2. assignFocus

```
DialogControl::assignFocus


>>--assignFocus-----------------------------------><
```

4.4.3. clearRect

```
DialogControl::clearRect
```

```

Form 1:

>>--clearRect(--rectangle--)---------------------><


Form 2:

>>--clearRect(--pt1--,--pt2--)-------------------><


Form 3:

>>--clearRect(--x-,--y-,--cx-,--cy--)------------><


Generic form:

>>--clearRect(--rectCoordinates--)---------------><
```

4.4.4. connectCharEvent

```
DialogControl::connectCharEvent


>>--connectCharEvent(--+--------------+--)-------><
                       +--methodName--+
```

4.4.5. connectFKeyPress

```
DialogControl::connectFKeyPress


>>--connectFKeyPress(--methodName--)-------------><
```

4.4.6. connectKeyPress

```
DialogControl::connectKeyPress


>>--connectKeyPress(--methodName--,--keys--+-----------+--)-----><
                                           +-,-filter--+
```

4.4.7. data

```
DialogControl::data


>>--data-----------------------------------------><
```

4.4.8. data=

```
DialogControl::data =
>>--data = newData-------------------------------><
```

4.4.9. disconnectKeyPress

```
DialogControl::disconnectKeyPress


>>--disconnectKeyPress(--+--------------+--)-----><
                         +--methodName--+
```

4.4.10. getTextSizeDlg

```
DialogControl::getTextSizeDlg


>>--getTextSizeDlg(--text--+-------------+--+-------------+--+---------+--)----><
                           +-,-fontname--+  +-,-fontSize--+  +-,-hwnd--+
```

4.4.11. group

```
DialogControl::group


>>--group(--+-----------+--)---------------------><
            +-wantStyle-+
```

4.4.12. hasKeyPressConnection

```
DialogControl::hasKeyPressConnection


>>--hasKeyPressConnection(--+--------------+--)--><
                            +--methodName--+
```

4.4.13. redrawRect

```
DialogControl::redrawRect


Form 1:

>>--redrawRect(--rectangle--+---------+--)-------><
                            +-,-erase-+

Form 2:

>>--redrawRect(--pt1--,--pt2--+---------+--)-----><
                              +-,-erase-+

Form 3:
```

```
>>--redrawRect(--x-,--y-,--x1-,--y1--+---------+--)-------------><
                                     +-,-erase-+

Generic form:

>>--redrawRect(--rectCoordinates--+---------+--)-><
                                  +-,-erase-+
```

4.4.14. setColor

```
DialogControl::setColor


>>--setColor(--bk--+-------+--)------------------><
                   +-,-fg--+
```

4.4.15. setSysColor

```
DialogControl::setSysColor


>>--setSysColor(--bk--+-------+--)---------------><
                      +-,-fg--+
```

4.4.16. tabStop

```
DialogControl::tabStop


>>--tabStop(--+-----------+--)-------------------><
              +-wantStyle-+
```

4.4.17. textSize

```
DialogControl::textSize


>>--textSize(--text--,--size--)------------------><
```

4.4.18. useUnicode

```
DialogControl::useUnicode


>>--useUnicode(--use--)--------------------------><
```

4.4.19. usingUnicode

```
DialogControl::usingUnicode
```

```
>>--usingUnicode--------------------------------><
```

4.5. DialogExtensions Mixin Class

The **`DialogExtensions`** class is a *mixin* class inherited by *PlainBaseDialog* class. Therefore the methods of this class are methods of every dialog. The methods of the **`DialogExtensions`** class are probably methods added after the original release of the ooDialog framework, thus the *extensions* part of the class name.

4.5.1. Method Table

The following table provides links to the documentation for the methods DialogExtensions Mixin Class:

Table 4.7. DialogExtensions Methods

Method	Description
addAutoStartMethod	Adds a method name and parameters to an internal queue whose items are started automatically and run concurrently when the dialog is executed.
changeBitmapButton	Changes the bitmap of a bitmap button.
clearControlRect	Clears the client area of the specified dialog control by redrawing the area with the background brush set to the default background color of a dialog.
clearRect	Clears the specified rectangular within the client area of a window.
clearWindowRect	Erases the client area of the given window.
createBrush	Creates a logical brush that has the specified style, color, and pattern.
determineSBPosition	Calculates and sets the new scroll bar position based on the position data retrieved from the scroll bar and the step information.
dimBitmap	Draws a bitmap step by step.
displaceBitmap	Sets the position of the bitmap within a bitmap button.
drawBitmap	Draws the bitmap for a bitmap button at the specified position.
drawButton	Draws the specified button.
endAsyncExecution	Used to end the asynchronous execution of a dialog started using the executedAsync method.
executeAsync	Creates the underlying Windows dialog, starts it executing, shows it in the same way as the execute method does, and returns immediately.
freeControlDC	Releases the device context of a control.
freeWindowDC	Releases the device context of a window.
getBitmapPosition	Retrieves the position, as a point, of the upper left corner of a bitmap within a bitmap button.
getBitmapSizeX	Retrieves the width of the bitmap that is set for a bitmap button.
getBitmapSizeY	Retrieves the height of the bitmap that is set for a bitmap button.
getBmpDisplacement	Retrieves the position of the bitmap within a bitmap button.
getControlDC	Returns the device context of a dialog control.
getControlRect	Returns the size and position rectangle of the specified dialog control.

Method	Description
getListItemHeight	Returns the height of the items in a list box in dialog units. **(Inaccurate)**
getListItemHeightPx	Returns the height of the items in a list box in pixels.
getListWidth	Returns the scrollable width of a list box in dialog units. **(Inaccurate)**
getListWidthPx	Returns the scrollable width of a list box in pixels.
getSBPos	Returns the current position of a scroll bar control.
getSBRange	Returns the range of a scroll bar control.
getWindowDC	Returns the device context of a window.
installAnimatedButton	Installs an animated button and runs it concurrently with the main activity.
installBitmapbutton	Connects bitmap(s) and a method with a push button.
moveControl	Moves a dialog control to another position within the dialog window.
popup	Starts a modeless dialog executing and returns immediately.
popupAsChild	Assigns a parent dialog, starts a modeless dialog executing, and returns immediately. The dialog is closed automatically when the parent dialog ends.
redrawControl	Redraws the specified dialog control.
redrawRect	Redraws a rectangle within the client area of this dialog, or optionally the client area of a specified window.
redrawWindow	Redraws the specified window.
redrawWindowRect	Forces this dialog to completely redraw its client area, or optionally the client area of a specified window.
resizeControl	Changes the size of a dialog control.
scrollBitmapFromTo	Scrolls the bitmap within the bitmap button from one position to another.
scrollButton	Moves a rectangle within the button and redraws the uncovered area with the button background color.
scrollInControl	Scrolls text in a dialog control using the specified font.
scrollText	Scrolls text in the specified window using the specified font.
setBitmapPosition	Sets the position of the upper left corner of a bitmap within the bitmap button.
setControlFont	Changes the font for specified dialog control.
setControlColor	Sets the background color, and optionally the text color, for the specified dialog control.
setControlSysColor	Sets the background color, and optionally the text color, for the specified dialog control using the system colors.
setForegroundWindow	Brings the specified window to the foreground.
setListColumnWidth	Sets the width of all columns in a list box in dialog units. **(Inaccurate)**
setListColumnWidthPx	Sets the width of all columns in a list box in pixels.
setListItemHeight	Sets the height for all items in a list box in dialog units. **(Inaccurate)**
setListItemHeightPx	Sets the height for all items in a list box in pixels.
setListWidth	Sets the scrollable width of a list box in dialog units. **(Inaccurate)**
setListWidthPx	Sets the scrollable width of a list box in pixels.

Method	Description
setSBPos	Sets the current position of a scroll bar control.
setSBRange	Sets the range of a scroll bar control.
setSBRange	Changes the size and position of the specified window.
writeToWindow	Writes text to the dialog or dialog control, specified by its window handle, in the given font at the specified position.
writeToControl	Writes text to the dialog control, specified by its resource ID, in the given font at the specified position.

4.5.2. addAutoStartMethod

```
>>--addAutoStartMethod(--+---------+--,--methodName--+---------------+--)------><
                         +-inClass-+                 +-,--parameters-+
```

The addAutoStartMethod method adds a method name and parameters to a special internal queue. All methods in this queue will be started automatically and run concurrently when the dialog is executed. The given method (MethodName) in the given class (InClass) is started concurrently with the dialog when the dialog is activated using the *execute* or *executeAsync* method. This is useful for processing animated buttons.

Arguments:

The arguments are:

inClass

The class where the method is defined. If this argument is omitted, the method is assumed to be defined in the dialog class.

methodName

The name of the method

parameters

All parameters that are passed to this method

Example:

The following example installs the ExecuteB method of the MyAnimatedButton class so that it is processed concurrently with the dialog execution:

```
MyDialog~addAutoStartMethod("MyAnimatedButton", "ExecuteB")

::class MyAnimatedButton
::method ExecuteB
     .
     .
     .
```

4.5.3. changeBitmapButton

```
>>--changeBitmapButton(--id-,-bN--+------+--+------+--+------+--+--------+--)--><
                                  +-,-bF-+  +-,-bS-+  +-,-bD-+  +-,-opts-+
```

Changes the bitmaps for an already *installBitmapbutton* bitmap button and optionally immediately redraws the button.

Arguments:

The arguments are:

id [required]

The resource ID for the bitmap button whose bitmaps are being changed. May be numeric or *symbolic*.

bN [required]

The bitmap to use for the normal state of the button. See the remarks for information on how the bitmap may be specified.

focused [optional]

The bitmap to use for the focused state of the button. See the remarks for information on how the bitmap may be specified. If this argument is omitted, the normal bitmap is used for the focused state.

selected [optional]

The bitmap to use for the selected state of the button. See the remarks for information on how the bitmap may be specified. If this argument is omitted, the normal bitmap is used for the selected state.

disabled [optional]

The bitmap to use for the disabled state of the button. See the remarks for information on how the bitmap may be specified. If this argument is omitted, the normal bitmap is used for the disabled state.

style [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant:

FRAME	INMEMORY	NODRAW
USEPAL	STRETCH	

FRAME

With this option, when the ooDialog framework draws the bitmap button, it attempts to draw it with the 3D effect. Note that the drawing method is outdated and will give a somewhat different appearance to the button than that of buttons in modern Windows versions.

USEPAL

Uses the colors of the bitmap for the normal state to create a system color palette. Use this option when the bitmap was created with a palette other than the default Windows color palette. This palette is used when drawing all of the bitmap states so when different bitmaps are used for the different states, they should all use the same colors.

INMEMORY

The bitmaps can be loaded into memory through the *loadBitmap* method. In this case, the programmer uses the bitmap handle to specify the bitmaps and **must** use the INMEMORY keyword.

STRETCH

If the size of the bitmap is smaller than the size of the button rectangle, the bitmap is stretched to match the size of the button. This option has no effect on bitmaps loaded from a dynamic-link library.

NODRAW

By default, when the button bitmaps are changed, the button is immediately redrawn. If this keyword is used, the button is not redrawn immediately. The next time the button needs to be redrawn, it will be redrawn using the new bitmaps.

Return value:

Returns 0 for success. On error returns -1 if the resource ID was symbolic and it could not be resolved to a numeric ID, or 1 for any other error.

Remarks:

The arguments specifying the bitmaps can be a bitmap file name, a bitmap *handle*, or the resource number of a bitmap compiled into the resource DLL of a *ResDialog*. The bitmap arguments must all be the same type. I.e. all file names, all resource IDs, or all handles. Symbolic resource IDs can be used for the bitmap arguments. If the bitmap arguments are handles the style argument must contain the INMEMORY keyword.

The bitmap button must have already been assigned bitmaps, if there are no assigned bitmaps this method fails. All of the existing assigned bitmaps are removed, the programmer must reassign all 4 bitmaps, even if the bitmap for some state is the same as the previous bitmap.

In ooDialog a bitmap button is an owner-drawn button that uses the supplied bitmap(s) for the button. ooDialog internally handles the drawing of the button.

Up to 4 bitmaps can be assigned to the button. These bitmaps are drawn for the different button states: normal, focused, selected and disabled. The normal bitmap must be supplied and that bitmap is used for any of the other states if no bitmap for the state is supplied.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

```
self~changeBitmap("IDOK", "AddBut_n.bmp", "AddBut_f.bmp", -
                  "AddBut_s.bmp", "AddBut_d.bmp", "FRAME")
```

4.5.4. clearControlRect

```
>>--clearControlRect(--id--)---------------------><
```

The *clearControlRect* clears the client area of the specified dialog control by redrawing the area with the background brush set to the default background color of a dialog.

Arguments:

The single argument is:

id [required]

The resource id of the dialog control that is to be cleared. May be numeric or *symbolic*.

Return value:

0 on success, or 1 on any error.

Remarks:

The client area of the control is not really *cleared*, but rather the dialog background is painted over the top of the area. Most dialog controls do not have a border, but if the control does have a border, the border is not cleared because it is outside of the client area.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

4.5.5. clearRect

```
Form 1:

>>--clearRect(--hwnd--,--rectangle--)------------><


Form 2:

>>--clearRect(--hwnd--,--pt1--,--pt2--)----------><


Form 3:

>>--clearRect(--hwnd--,--x--,--y--,--x1--,--y1--)---------------><


Generic form:

>>--clearRect(--hwnd--,--rectCoordinates--)------><
```

The *clearRect* method clears the specified bounding *rectangle* within the client area of a window. The rectangle is *cleared* by redrawing it with the background brush set to the typical background color for dialog boxes and three dimensional elements.

Arguments:

The arguments are:

hwnd [required]

The *handle* of the window containing the rectangle to be cleared.

rectCoordinates [required]

The coordinates of the rectangle to be cleared. The coordinates specify the upper left and lower right corners of the rectangle. The corners can be specified using either a *Rect* object, two *Point* objects, or the individual x and y coordinates of each corner.

The coordinates are specified as *client area* coordinates, in pixels.

Return value:

0 on success, 1 on error.

Remarks:

The rectangle is not really cleared, but rather is redrawn using the normal background color for a dialog box.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example is a complete program, it can be copy and pasted into a file and executed. Running the program and reading the code should give a better understanding of the *clearRect* method than a text description alone.

```
/* Shows clearRect() at work. */

  dlg = .Simple~new
  dlg~execute("SHOWTOP")

  return 0
-- End of entry point.

::requires "ooDialog.cls"

::class 'Simple' subclass UserDialog

::method init

  forward class (super) continue
  self~createCenter(245, 132, "clearRect() Example")

::method defineDialog

  self~createGroupBox(100, 10, 13, 107, 70, , "Test Group Box")
  self~createListView(200, 130, 13, 107, 70, "REPORT")

  self~createPushButton(300, 115, 108, 65, 14, , "Clear Group Box", onClear)
  self~createPushButton(IDOK, 185, 108, 50, 14, "DEFAULT", "Ok")

::method initDialog
  expose list gb pb

  list = self~newListView(200)

  list~insertColumn(0, "Text", 40)
  list~insertColumn(1, "Number", 30)
  do i = 1 to 3
    list~addRow(i, , "Line", i)
  end

  gb = self~newGroupBox(100)
  pb = self~newPushButton(300)

::method onClear unguarded
  expose list gb pb

  text = pb~getText
  select
    when text == "Clear Group Box" then do
      text = "Clear List View"
      r = gb~clientRect
      r~right %= 2
      r~bottom %= 2
      self~clearRect(gb~hwnd, r)
    end
    when text == "Clear List View" then do
      text = "Refresh"
      self~clearRect(list~hwnd, 10, 10, 55, 80)
    end
    otherwise do
      text = "Clear Group Box"
      gb~redrawClient
      list~redrawClient
    end
  end
  -- End select

  pb~setText(text)
```

4.5.6. clearWindowRect

```
>>--clearWindowRect(--hwnd--)--------------------><
```

The clearWindowRect method erases the client area of the given window.

Arguments:

The only argument is:

hwnd

The handle of the window. See *getSelf*, *get*, or *getControlHandle* for some methods to get a window handle.

Example:

The following example gets the window handle of a dialog control with the symbolic resource ID of IDC_PB_DONE and then clears the dialog control window:

```
hwnd = self~getControlHandle(IDC_PB_DONE)
MyDialog~clearWindowRect(hwnd)
```

4.5.7. createBrush

```
>>--createBrush(--+---------+--+-------------------+--)------------------------><
                  +--color--+  +-,-brushSpecifier--+
```

The *createBrush* method creates a logical brush that has the specified style, color, and pattern.

Arguments:

Both arguments are optional. If both arguments are omitted then a hollow brush is created. Otherwise, the arguments are:

color [optional]

The *color number*. When this argument is omitted and *brushSpecifier* is used, the color number defaults to 1.

brushSpecifier [optional]

If this argument is omitted a solid brush using the color specified is created. Otherwise, this argument can be the name of a bitmap file, the resource ID of a bitmap compiled in the resource only DLL of a *ResDialog*, or one of the following keywords. The keywords create a hatched brush. A bitmap file name will cause the bitmap to be loaded into memory and then used for the brush. Case is not significant in the keywords.

UPDIAGONAL	DOWNDIAGONAL
CROSS	HORIZONTAL
DIAGCROSS	VERTICAL

UPDIAGONAL
: A 45-degree upward, left-to-right hatch.

CROSS
: A horizontal and vertical crosshatch.

DIAGCROSS
: A 45-degree crosshatch.

DOWNDIAGONAL
: A 45-degree downward, left-to-right hatch.

HORIZONTAL
: A horizontal hatch.

VERTICAL
: A vertical hatch.

Remarks:

A brush is a bitmap that the operating system uses to paint the interiors of filled shapes. After the programmer creates a brush, it can then be selected into a device context using the *objectToDC* method. When the brush is no longer needed use the *deleteObject* method to release the operating resources used by the brush.

When *brushSpecifier* is a non-negative whole number, it is taken to be the resource ID of a bitmap compiled in to the resource only DLL of a **`ResDialog`**. If the dialog is not a **`ResDialog`**, or if there is no matching bitmap in the DLL, the method fails. Note that, technically, if the dialog is not a **`ResDialog`**, the **`ooDialog.dll`** is searched for the bitmap. However, since there are currently no bitmaps compiled in to **`ooDialog.dll`**, the restriction that the dialog must be a **`ResDialog`** is essentially correct.

The *createBrush* method here is almost identical to the *createBrush* method of the *WindowsExtensions* class. The method documented here is a method of the *DialogExtensions* class and is therefore inherited only by the *dialog* object.

In the dialog object, the *createBrush* method of the **`DialogExtensions`** class over-rides the *createBrush* method of the **`WindowExtensions`** class. Therefore, this documentation is essentially the dialog object's *createBrush* documentation. The *createBrush* documentation is for the dialog control object's *createBrush* method.

Details:

Sets the *.systemErrorCode*.

4.5.8. determineSBPosition

```
>>--determineSBPosition(--id--,--posdata--+-----------+--+---------+--)--------><
                                          +-,-single--+  +-,-page--+
```

The determineSBPosition method calculates and sets the new scroll bar position based on the position data retrieved from the scroll bar and the step information.

Protected:

This method is protected.

Arguments:

The arguments are:

id

The ID of the scroll bar.

posdata

The position information sent with the connected scroll bar event.

single

This number is added (or subtracted if negative) to the current position for a single step. If omitted, the single step size is 1.

page

This number is added (or subtracted if negative) to the current position for a page step. If omitted, the page step size is 10.

Return value:

The new scroll bar position.

Example:

The following example demonstrates how to update the scroll bar position. Each time the ScrollBarEventHandler is called by an event for scroll bar SB_SIZE, the position of the scroll bar is calculated and updated. **`posdata`** is sent along with the scroll bar event.

```
   /* Method ScrollBarEventHandler is connected to item SB_SIZE */
::method ScrollBarEventHandler
    use arg posdata, sbwnd
    pos = self~determineSBPosition("SB_SIZE",posdata,1,25)
    return pos
```

4.5.9. dimBitmap

```
>>--dimBitmap(--id-,-bmp-,-cx-,-cy--+---------+--+---------+--+---------+--)---><
                                    +-,-stepX-+  +-,-stepY-+  +-,-steps-+
```

Draws a bitmap on the client area of a button control step by step.

Arguments:

The arguments are:

id [required]

The resource ID of the button. May be numeric or symbolic.

bmp [required]

A handle to the bitmap loaded with *loadBitmap*.

cx, cy [required]

The width and height of the bitmap.

stepx, stepy [optional]

The number of pixels to increment the x and y position of the bitmap at each step. The default is 2 pixels for both *cx* and *cy*.

steps [optional]

The number of iterations used to draw the bitmap. The bitmap is redrawn at each step. The default is 10.

Return value:

Returns 0 on success and -1 if an error is detected.

Remarks:

This method is only intended to be used with button controls. It will fail if the resource ID argument specifies a dialog control that is not a button.

The *underlying* Windows dialog must exist to use this method.

Details:

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

4.5.10. displaceBitmap

```
>>--displaceBitmap(--id--,--x--,--y--)-----------><
```

The *displaceBitmap* method sets the position of a bitmap within the *client area* of a bitmap button.

Arguments:

The arguments are:

x [required]

The x coordinate of the top left corner of the bitmap within the client area of the bitmap button.

y [required]

The y coordinate of the upper left corner of the bitmap within the client area of the bitmap button.

Remarks:

Both the x and y coordinates are in pixels and can have negative values.

Recall that client area coordinates are specified in relation to the (0, 0) point of the client area of a window. Negative values then place the bitmap outside of the client area of the bitmap

The *setBitmapPosition* method is most likely a better method to use than the *displaceBitmap*. It accepts either a **`Point`** object or the individual x and y coordinates to specify the position of the bitmap. *Set position* more accurately describes what the method does than *displace*.

Example:

The following example moves the bitmap within the associated bitmap button 4 screen pixels to the right and 3 pixels upward. Contrast this with the setBitmapPosition *example*:

```
parse value self~getBmpDisplacement(IDC_PB_BITMAP1) with dx dy
if dx <> -1 then do
  dx += 4
  dy -=3
  self~displaceBitmap(IDC_PB_BITMAP1, dx, dy)
end
```

4.5.11. drawBitmap

```
>>--drawBitmap(--+--------+--,--id--+------+--+------+--+---------+------------>
                 +--hwnd--+         +-,-x--+ -+-,-y--+ -+-,-bmpX--+

>--+---------+--+----------+--+-----------+--)--------------------------------><
   +-,-bmpY--+  +-,-width--+  +-,-height--+
```

Draws all, or part of, the bitmap for a bitmap button at the specified position.

Arguments:

The arguments are:

hwnd [optional]

The window *handle* of the bitmap button. This argument is, and has always been, ignored. It is simply retained for backwards compatibility.

id [required]

The resource ID of the bitmap button. May be numeric or *symbolic*.

x, y [optional]

The client area *coordinates* where the upper left corner of the bitmap is to be drawn, in pixels. The default is (0, 0).

bmpX, bmpY [optional]

The coordinates in the bitmap for the upper left corner of the portion that is to be drawn. The default is (0, 0). This, along with the *width* and *height* arguments, can be used to draw only a portion of the bitmap. For instance, to draw only the lower right quadrant of the bitmap, this argument could be set to the mid point of the bitmap.

width [optional]

The width, starting from ()*bmpX*, *bmpY*) of the bitmap of the portion for the bitmap being drawn. If omitted or 0, the remaining width of the bitmap is drawn.

height [optional]

The height, starting from ()*bmpX*, *bmpY*) of the bitmap for the portion of the bitmap being drawn. If omitted or 0, the remaining height of the bitmap is drawn.

Return value:

Returns 0 on success or 1 for error.

Remarks:

This method is only for bitmaps that have been assigned to a bitmap button through the *installBitmapbutton* method. If there is no bitmap assigned to the button then 1 is returned.

Unlike the *setBitmapPosition* method, which permanently sets the position for the bitmap, the *drawBitmap* immediately draws the bitmap, or part of it, at the specified position. If the button needs to be redrawn, maybe because the window was covered and then uncovered, the bitmap is drawn at the position set with *setBitmapPosition*. *drawBitmap* is used by *scrollBitmapFromTo*, for example.

You can use the *drawBitmap* method to animate a bitmap by providing a bitmap that contains several images and use the offset and extension arguments to display a single image of the bitmap.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

4.5.12. drawButton

```
>>--drawButton(--id--)---------------------------><
```

The drawButton method draws the given button.

Arguments:

The only argument is:

id

The ID of the button

4.5.13. endAsyncExecution

```
>>--endAsyncExecution----------------------------><
```

The *endAsyncExecution* method is used to end the asynchronous execution of a dialog started using the *executeAsync* method. The *endAsyncExecution* method does not return until the user closes the dialog.

Arguments:

The method does not take any arguments.

Return value:

The return values are the same as for the *execute* method and are:

0

Some error occurred, the dialog was not executed.

1

The user terminated the dialog using an ok command.

2

The user terminated the dialog using a cancel command.

Remarks:

Every dialog that is started using the *executeAsync* method must have a matching *endAsyncExecution* invocation. This is the only way to ensure the proper termination of the dialog. This does not apply of course if the invocation of *executeAsync* failed.

Example:

The executeAsync *example* shows the proper use of the *endAsyncExecution* method.

4.5.14. executeAsync

```
>>--executeAsync(--+-----------+--+---------------+--+----------+--)----------><
```

```
                    +--ignored--+  +-,--showOption--+  +-,--icon--+
```

The *executeAsync* method creates the *underlying* Windows dialog, starts it executing, and shows it in the same way as the *execute* method does. However, *executeAsync* does not wait for the dialog to be closed, rather it returns immediately. This allows the programmer to execute other Rexx statements asynchronously with the dialog execution. The *endAsyncExecution* method is used to halt the asynchronously execution and wait for the dialog to be closed.

Arguments:

The arguments are:

ignored [optional]

This argument is completely ignored. It is a hold over from older versions of ooDialog, versions prior to 4.0.0. However, for backwards program compatibility the argument must be returned. The programmer should omit the argument altogether, or could put in any value as a place holder.

showOption [optional]

Zero or one of the following keywords to specify how the dialog is shown. This keyword is used in the automatic invocation of the *show* method. Case is not significant. If this argument is omitted, the NORMAL keyword is used:

NORMAL

Makes the dialog visible in its default position and window size. This has the effect of restoring the dialog size and position if it is minimized or maximized. This is the default if the argument is omitted.

DEFAULT

DEFAULT is an alias for NORMAL. The two keywords are functionally identical.

SHOWTOP

Makes the dialog visible and the topmost window.

HIDE

Makes the dialog invisible.

MIN

Minimizes the dialog and activates the next window in the window order.

MAX

Maximizes, and makes visible if necessary, the dialog.

INACTIVE

Makes the dialog visible without changing the active window. When the NORMAL keyword is used, the dialog is shown and becomes the active window. The INACTIVE keyword makes the dialog visible without changing the focus from the current active window.

RESTORE

Makes the dialog visible and restores it to its original size and position if it was minimized or maximized. An application should specify this flag when restoring a minimized window.

icon [optional]

The resource ID of the *dialog icon*. May be numeric or *symbolic*. If an icon ID is not supplied, the default ooDialog icon will be used.

Return value:

The return values are:

0

The dialog was created and shown without problems. The dialog is currently executing.

1

An error occurred, the dialog was not created and is not executing. Do not invoke *endAsyncExecution* in this case.

Remarks:

Although any of the show keywords can be used, for the *executeAsync* method the SHOWTOP, HIDE, and MAX keywords make the most sense. SHOWTOP is the usual keyword. Rather than use the HIDE keyword, it is more practical to simply create the dialog **without** the VISIBLE style.

If another ooDialog dialog has been started by the Rexx program, it is disabled when *executeAsync* is invoked. This in effect makes the dialog a *modal* dialog. To start a *modeless* dialog use the *popup* or *popupAsChild* methods.

Example:

This example shows an application that creates a *tool palette* dialog. The tool palette dialog can not be executed until its underlying owner dialog has been created. So the main dialog is started using *executeAsync*, then the palette dialog is executed, and finally *endAsyncExecution* is used to wait for the main dialog to close:

```
.application~useGlobalConstDir("O", "useTools.h")

dlg = .MainDialog~new

dlgTool = .ToolPaletteDlg~new
dlgTool~ownerDialog = dlg

dlg~executeAsync("SHOWTOP", IDI_DLG_OOREXX)

dlgTool~popup("SHOWTOP")

dlg~endAsyncExecution
```

4.5.15. freeControlDC

```
>>--freeControlDC(--id--,--dc--)-----------------><
```

The freeControlDC method releases the device context of a control.

Arguments:

The arguments are:

id

The resource ID of the control.

dc

The device context previously received by the *getControlDC* method

4.5.16. freeWindowDC

```
>>--freeWindowDC(--hwnd--,--dc--)----------------><
```

The freeWindowDC method frees the device context of a window.

Arguments:

The arguments are:

hwnd

The window handle

dc

The device context previously received by the *getWindowDC* method

4.5.17. getBitmapPosition

```
>>--getBitmapPosition(--id--,--pos--)------------><
```

Retrieves the position in client *coordinates* of the upper left corner of a bitmap within a bitmap button.

Arguments:

The arguments are:

id [required]

The resource ID of the bitmap button. May be numeric or *symbolic*.

pos

A *Point* object. On a successful return the point object is updated with the position of the bitmap. On an error return, the point object is left unchanged.

Return value:

Returns **`.true`** on success and **`.false`** on error.

Remarks:

Recall that client area coordinates are relative to the (0, 0) point of the client area of a window. Both or either of the coordinates can be negative. A negative x will place the bitmap to the left of the edge of the button. A negative y will place the bitmap above the top of the bitmap.

This method is only for bitmaps that have been assigned to a bitmap button through the *installBitmapbutton* method. If there is no bitmap assigned to the button then **`.false`** is returned.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

4.5.18. getBitmapSize

```
>>--getBitmapSize(--id--)------------------------><
```

Retrieves the size of the bitmap for a bitmap button as a *Size* object.

Arguments:

The single arguments is:

id [required]

The resource ID of the bitmap button. May be numeric or *symbolic*.

Return value:

On success, returns the bitmap size. On failure returns -1.

Remarks:

If *id* is not the resource ID of a bitmap button, or if the bitmap for the button can not be located, the negative one is returned.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example retrieves the size of the bitmap for the bitmap button with symbolic resource ID of IDC_PB_TICKET:

```
size = self~getBitmapSize(IDC_PB_TICKET)
say 'Size for the ticket push button bitmap:' size~width 'by' size~height

/* Output might be:

   Size for the ticket push button bitmap: 152 by 178

*/
```

4.5.19. getBitmapSizeX

```
>>--getBitmapSizeX(--id--)-----------------------><
```

Gets the width of the bitmap assigned to this bitmap button in pixels.

Arguments:

The single argument is:

id [required]

The resource ID of the bitmap button. May be numeric or *symbolic*.

Return value:

On success, the width of the bitmap for the bitmap button in pixels, on error -1.

Remarks:

This method is only for bitmaps that have been assigned to a bitmap button through the *installBitmapbutton* method. If there is no bitmap assigned to this button then -1 is returned.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

4.5.20. getBitmapSizeY

```
>>--getBitmapSizeY(--id--)-----------------------><
```

Gets the height of the bitmap assigned to this bitmap button in pixels.

Arguments:

The single argument is:

id [required]

The resource ID of the bitmap button. May be numeric or *symbolic*.

Return value:

On success, the height of the bitmap for the bitmap button in pixels, on error -1.

Remarks:

This method is only for bitmaps that have been assigned to a bitmap button through the *installBitmapbutton* method. If there is no bitmap assigned to this button then -1 is returned.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

4.5.21. getBmpDisplacement

```
>>--getBmpDisplacement(--id--)-------------------><
```

Returns a string containing the *client area* coordinates of the bitmap in a bitmap button.

Arguments:

The single argument is:

id [required]

The resource ID of the bitmap button. May be numeric or *symbolic*.

Return value:

A string with the X position and then the Y position of the bitmap, in pixels. The X and Y coordinates are separated by a blank. On error, -1 is returned.

Remarks:

Recall that client area coordinates are relative to the (0, 0) point of the client area of a window. Both or either of the coordinates can be negative.

This method is only for bitmaps that have been assigned to a bitmap button through the *installBitmapbutton* method. If there is no bitmap assigned to the button then -1 is returned.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This following example shows how to use the *getBmpDisplacement* method:

```
bmpPos = self~getBmpDisplacement(IDC_PB_TICKET)
parse var bmpPos x y
```

4.5.22. getControlDC

```
>>--getControlDC(--id--)-------------------------><
```

The getControlDC method returns the device context of a dialog control. Do not forget to free the device context after you have completed the operations (see *freeControlDC*).

Arguments:

The only argument is:

id

The resource ID of the dialog control. May be numeric or symbolic.

4.5.23. getControlRect

```
>>--getControlRect(--id--)-----------------------><
```

The getControlRect method returns the size and position rectangle of the given dialog control. The four values (left, top, right, bottom) are returned in one string separated by blanks.

Arguments:

The only argument is:

id

The ID of the dialog control.

4.5.24. getListItemHeight

```
>>--getListItemHeight(--id--)--------------------><
```

The *getListItemHeight* method returns the height of the items in a list box, in dialog units.

The value is usually *inaccurate* because it is calculated using *factorX*. The programmer is advised to use the *getListItemHeightPx* method instead, which will return the correct height of a list item.

Arguments:

The only argument is:

id

The ID of the list box of which you want to know the item height.

Return value:

The height of the list box items, in dialog units.

4.5.25. getListItemHeightPx

```
>>--getListItemHeightPx(--id--)------------------><
```

Gets the height, in pixels, of an individual item in a *ListBox* control.

Arguments:

The single arguments is:

id [required]

The resource ID of the list box. May be numeric or *symbolic*.

Return value:

Returns the height of each item in the list box, in pixels.

4.5.26. getListWidth

```
>>--getListWidth(--id--)-------------------------><
```

The getListWidth method returns the scrollable width of a list box, in dialog units.

The value is usually *inaccurate* because it is calculated using *factorX*. The programmer is advised to use the *getListWidthPx* method instead, which will return the correct width in pixels.

Arguments:

The only argument is:

id

The ID of the list box of which you want to know the scrollable width.

Return value:

The width of the scrollable area of the list box, in dialog units.

4.5.27. getListWidthPx

```
>>--getListWidthPx(--id--)-----------------------><
```

Retrieves the width, in pixels, that a list box can be scrolled horizontally, if the list box has a horizontal scroll bar.

Arguments:

The single arguments is:

id [required]

The resource ID of the list box. May be numeric or *symbolic*.

Return value:

The scrollable width of the list box, in pixels

4.5.28. getSBPos

```
>>--getSBPos(--id--)-----------------------------><
```

The getSBPos method returns the current position of a scroll bar control.

Arguments:

The only argument is:

id

The ID of the scroll bar.

4.5.29. getSBRange

```
>>--getSBRange(--id--)---------------------------><
```

The getSBRange method returns the range of a scroll bar control. It returns the two values (minimum and maximum) in one string, separated by a blank.

Protected:

This method is protected.

Arguments:

The only argument is:

id

The ID of the scroll bar.

Example:

The following example demonstrates how to get the minimum and the maximum values of the scroll bar:

```
      .
      .
      .
::method dumpSBRange

  sbRange = self~getSBRange(IDC_SB)
  parse var sbRange sbMin sbMax
  say sbMin " - " sbMax
```

4.5.30. getWindowDC

```
>>--getWindowDC(--hwnd--)------------------------><
```

The getWindowDC method returns the device context of a window. Do not forget to free the device context after you have completed the operations (see *freeWindowDC*).

Arguments:

The only argument is:

hwnd

The handle of the window

4.5.31. installAnimatedButton

```
>>--installAnimatedButton(--id-,-+--------+-+--------+-,-bmpF-+--------+-,--mX->
                                 +-,-mth--+ +-,-cls--+        +-,-bmpT-+

>--,-mY--+---------+--+---------+--,--delay---+--------+--+--------+--)--------><
         +-,-sizeX-+  +-,-sizeY-+             +-,-xNow-+  +-,-yNow-+
```

The installAnimatedButton method installs an animated button and runs it concurrently with the main activity.

Arguments:

The arguments are:

id

The ID of the button

mth

The name of a method within the same class. This method is called each time the button is clicked.

cls

The class that controls the animation. The default is *AnimatedButton* Class.

bmpF

The ID of the first bitmap in the animation sequence within a binary resource. It can also be the name of an array stored in the .local directory containing handles of bitmaps to be animated and bmpTo is omitted. See *loadBitmap* for how to get bitmap handles. The array starts at index 1.

bmpT

The ID of the last bitmap in the animation sequence within a binary resource. If omitted, bmpFrom is expected to be the name of an array stored in .local that holds the bitmap handles of the bitmaps that are to be animated.

mX, mY

Size of one move (in pixels)

sizeX, sizeY

Size of the bitmaps (if omitted, the size of the bitmaps is retrieved)

delay

The time in milliseconds the method waits after each move

xNow, yNow

The starting position of the bitmap

Example:

The following example defines and runs an animated button. The example loads ten bitmaps ("anibmp1.bmp" to "anibmp10.bmp") into memory and stores them into the array "My.Bitmaps" that is stored in the .local directory. The name "My.Bitmaps" is specified as the **`bmpfrom`** and **`bmpto`** is omitted. After the dialog execution the bitmaps are removed from memory again. The sample also uses a different animation class (".MyAnimation") which subclasses from .AnimatedButton and overrides method HitRight which plays a tune each time the animated bitmap hits the right border.

```
/* store array in .local */
.Local["My.Bitmaps"] = .array~new(10)
/* load 10 bitmaps into .local array */
do i= 1 to 10
  .Local["My.Bitmaps"][i] = Dialog~loadBitmap("anibmp"i".bmp")
  /* you could also use .My.Bitmaps[i] =  ... */
end

/* connect bitmap sequence and .MyAnimated class with button IDANI */
Dialog~installAnimatedButton("IDANI", ,.MyAnimation,"My.Bitmaps", ,1,1, , ,100)

...
Dialog~execute
...

/* Free the bitmap previously loaded */
do bmp over .Local["My.Bitmaps"] /* You could also use do bmp over .My.Bitmaps */
   Dialog~removeBitmap(bmp)
end
```

```
::class MyAnimation subclass AnimatedButton

/* play sound.wav whenever the bitmap hits the right border */

::method HitRight
   ret = Play("sound.wav", yes)
   return self~super:hitright
```

4.5.32. installBitmapbutton

```
>>--installBitmapButton(-id-+-------+-,-bN--+------+-+------+-+-------+-+--------+-)-><
                            +-,-mth-+       +-,-bF-+ +-,-bS-+ +-,-bD--+ +-,-opts-+
```

The *installBitmapButton* method installs the bitmaps for a bitmap push button and optionally connects the push button click *event* with a method in the Rexx dialog. When the button needs to be drawn, the ooDialog framework draws the button using the supplied bitmaps rather than having the operating system draw a default push button.

Arguments:

The arguments are:

id [required]

The resource ID of the button. May be numeric or *symbolic*.

mth [optional]

The name of the event handler method in the Rexx dialog that will be invoked when the push button is clicked. If this method is omitted, then the event notification is not connected.

bN [required]

The source of the bitmap to be used for the button face when it is in the *normal* state. See the Remarks section for details on how to specify a bitmap source for this method.

bF [optional]

The source of the bitmap to be used for the button face when it is in the *focused* state. See the Remarks section for details on how to specify a bitmap source for this method.

bS [optional]

The source of the bitmap to be used for the button face when it is in the *selected* state. See the Remarks section for details on how to specify a bitmap source for this method.

bD [optional]

The source of the bitmap to be used for the button face when it is in the *disabled* state. See the Remarks section for details on how to specify a bitmap source for this method.

opts [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant:

FRAME	STRETCH
INMEMORY	USEPAL

FRAME

Draws a frame around the button. When this option is used, the appearance of the bitmap button is more similar to a normal Windows button, except that the button face is drawn using the bitmap rather than text.

INMEMORY

This option *must* be used if the named bitmaps are already loaded into memory by using the *loadBitmap* method. The bitmap source arguments must be specified as the bitmap *handle* returned from the *loadBitmap* method.

STRETCH

If this option is specified and the extent of the bitmap is smaller than the extent of the button rectangle, the bitmap is adapted to match the extent of the button. STRETCH has no effect for bitmaps loaded through a DLL.

USEPAL

Stores the colors of the bitmap file as the system color palette. This option is needed when the bitmap was created with a palette other than the default Windows color palette. Use it for one button only, because only one color palette can be active at any time. *USEPAL* is invalid for a bitmap loaded from a DLL.

Return value:

Returns 0 on success. Otherwise, returns -1 if the *id* argument was symbolic and could not be resolved and 1 for any other error.

Remarks:

There are 3 ways to specify the bitmap source for the *bN*, *bf*, *bS*, and *bD* arguments. The arguments can be a bitmap file name, in which case the bitmap is loaded from the file, the arguments can be the resource ID of bitmap compiled into the resource DLL for a *ResDialog*, or the arguments can be a bitmap *handle* returned from the *loadBitmap* method.

However, *all* the bitmap source arguments **must** be the same type. I.e., if the *bN* argument is specified as a file name, if any of the other bitmap source arguments are used they must also be specified as file names. Symbolic IDs can be used when using resource IDs to specify the bitmap sources.

Once a bitmap push button is installed, the *changeBitmapButton* method can be used to change the bitmaps associated with the button.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example installs a bitmap push button using four bitmaps and connects the button click event to a method:

```
::method initDialog
  ...

  self~installBitmapButton(204, "onBmpButtonClicked",                    -
                              "AddBut_n.bmp", "AddBut_f.bmp",            -
                              "AddBut_s.bmp", "AddBut_d.bmp", "FRAME")
  ...

::method onBmpButtonClicked
  ...
```

4.5.33. moveControl

```
>>--moveControl(--id--,--xPos--,--yPos--+------------+--)------><
```

```
                                          +-,-showOpt--+
```

The moveControl method moves a dialog control to another position within the dialog window.

Arguments:

The arguments are:

id
    The ID of the dialog control you want to move

xPos, yPos
    The new position in dialog units relative to the dialog window

showOpt
    This argument can be one of the following keywords:

    HIDEWINDOW
        Hides the dialog

    SHOWWINDOW
        Shows the dialog

    NOREDRAW
        Moves the dialog control without updating the display. Use the *update* method to manually update the display.

4.5.34. popup

```
>>--popup(--+--------------+--+------------+--+--------+--)-----><
            +--showOption--+  +-,-ignored--+  +-,-icon-+
```

The *popup* method starts a *modeless* dialog executing and returns immediately.

Arguments:

The arguments are:

showOption [optional]
    Zero or one of the following keywords to specify how the dialog is shown. This keyword is used in the automatic invocation of the *show* method. Case is not significant. If this argument is omitted, the NORMAL keyword is used:

    NORMAL
        Makes the dialog visible in its default position and window size. This has the effect of restoring the dialog size and position if it is minimized or maximized. This is the default if the argument is omitted.

    DEFAULT
        DEFAULT is an alias for NORMAL. The two keywords are functionally identical.

    SHOWTOP
        Makes the dialog visible and the topmost window.

    HIDE
        Makes the dialog invisible.

    MIN
        Minimizes the dialog and activates the next window in the window order.

MAX

Maximizes, and makes visible if necessary, the dialog.

INACTIVE

Makes the dialog visible without changing the active window. When the NORMAL keyword is used, the dialog is shown and becomes the active window. The INACTIVE keyword makes the dialog visible without changing the focus from the current active window.

RESTORE

Makes the dialog visible and restores it to its original size and position if it was minimized or maximized. An application should specify this flag when restoring a minimized window.

ignored [optional]

This argument is completely ignored. It is a hold over from the original ooDialog implementation. However, since the *icon* argument follows it, the *ignored* must be retained. It serves as a place holder argument to preserve backwards compatibility. The programmer should simply omit it, or use a value such as **1** to serve as the place holder.

icon [optional]

The resource ID of the *dialog icon*. May be numeric or *symbolic*. If an icon ID is not supplied, the default ooDialog icon will be used.

Return value:

The return is a Rexx **Message** object, (see the Open Object Rexx reference manual to learn more about the **Message** class.) The message object can be used, for instance, to check if the dialog has been closed by the user, to obtain the result value from the execution of the dialog, etc..

Remarks:

Note that the example shown below for the *popup* method makes use of the returned **Message** object to illustrate a possible use of the return. However, it is a contrived use case. Typically it is easier to just work with the instantiated dialog object directly. (In the case of the example this would be the *dlg* variable.)

Example:

This example demonstrates how a dialog started with *popup* executes asynchronously to the code in the main portion of the program. The example code below is complete. I can be cut and pasted into a file and will run as is.

```
  dlg = .ExampleDlg~new( , "simple.h")
  msgObj = dlg~popup("SHOWTOP", IDI_DLG_OOREXX)

  do i = 1 to 1000
    say "Iteration" i
    j = msSleep(1000)

    if msgObj~completed then do
      val = msgObj~result
      select
        when val == 1 then say 'The user closed the dialog with Ok.'
        when val == 2 then say 'The user closed the dialog with Cancel.'
        otherwise say 'The dialog has closed with an error'
      end
      leave
    end
  end

  if \ msgObj~completed then say 'Iteration loop ended, dialog is still running.'

::requires "ooDialog.cls"
```

```
::class 'ExampleDlg' subclass UserDialog

::method init
  forward class (super) continue
  self~create(30, 30, 257, 123, "Simple Dialog", "CENTER")

::method defineDialog

  self~createPushButton(IDOK, 142, 99, 50, 14, "DEFAULT", "Ok")
  self~createPushButton(IDCANCEL, 197, 99, 50, 14, , "Cancel")
```

4.5.35. popupAsChild

```
>>--popupAsChild(--parent--+---------------+--+------------+--+---------+--)---><
                           +-,-showOption--+  +-,-ignored--+  +-,-icon--+
```

The *popupAsChild* method starts a *modeless* dialog executing and returns immediately. It is very similar to the *popup* method, but has the additional functionality that it is closed automatically when the *parent* dialog is closed.

Arguments:

The arguments are:

parent [required]
: Some other Rexx dialog object. A relationship is established with the *parent* dialog such that when the *parent* dialog is closed, this dialog is automatically closed also.

showOption [optional]
: Zero or one of the following keywords to specify how the dialog is shown. This keyword is used in the automatic invocation of the *show* method. Case is not significant. If this argument is omitted, the NORMAL keyword is used:

NORMAL
: Makes the dialog visible in its default position and window size. This has the effect of restoring the dialog size and position if it is minimized or maximized. This is the default if the argument is omitted.

DEFAULT
: DEFAULT is an alias for NORMAL. The two keywords are functionally identical.

SHOWTOP
: Makes the dialog visible and the topmost window.

HIDE
: Makes the dialog invisible.

MIN
: Minimizes the dialog and activates the next window in the window order.

MAX
: Maximizes, and makes visible if necessary, the dialog.

INACTIVE

Makes the dialog visible without changing the active window. When the NORMAL keyword is used, the dialog is shown and becomes the active window. The INACTIVE keyword makes the dialog visible without changing the focus from the current active window.

RESTORE

Makes the dialog visible and restores it to its original size and position if it was minimized or maximized. An application should specify this flag when restoring a minimized window.

ignored [optional]

This argument is completely ignored. It is a hold over from the original ooDialog implementation. However, since the *icon* argument follows it, the *ignored* argument must be retained. It serves as a place holder argument to preserve backwards compatibility. The programmer should simply omit it, or use a value such as **`1`** to serve as the place holder.

icon [optional]

The resource ID of the *dialog icon*. May be numeric or *symbolic*. If an icon ID is not supplied, the default ooDialog icon will be used.

Return value:

The return is a Rexx **`Message`** object, (see the Open Object Rexx reference manual to learn more about the **`Message`** class.) The message object can be used, for instance, to check if the dialog has been closed by the user, to obtain the result value from the execution of the dialog, etc..

Remarks:

Note that the relationship between the dialog started with *popupAsChild* and the *parent* dialog is one-way. When the *parent* dialog is closed, the dialog started with the *popupAsChild* method is closed. But, when the dialog started with *popupAsChild* is closed, the *parent* dialog continues to execute as normal. It is not effected by the closing of the *child* dialog.

Note also that the example shown below for the *popupAsChild* method makes use of the returned **`Message`** object from the *popup* method to illustrate a possible use of the return. However, it is a contrived use case. Typically it is easier to just work with the instantiated dialog object directly. (In the case of the example this would be the *parentDlg* variable.)

Example:

This example creates and executes a *parent* dialog. It then executes a second dialog, using *popupAsChild* to start the second dialog. The main program path then runs a loop to demonstrate that both dialogs are executing asynchronously. When the loop counter reaches 20, the program closes the parent dialog. This will also close the child dialog.

This example program is complete. It can be copy and pasted into a file and will execute as is:

```
parentDlg = .SimpleParentDlg~new

msgObj = parentDlg~popup("SHOWTOP", , IDI_DLG_OOREXX)

childDlg = .ExampleChildDlg~new

childDlg~popupAsChild(parentDlg, 'SHOWTOP', IDI_DLG_OODIALOG)

do i = 1 to 1000
  say "Iteration" i
  j = msSleep(1000)

  -- Check if the user closed the parent dialog.  We do
  -- not care if the user closed the child dialog, the
  -- parent will keep executing.
```

```
    if msgObj~completed then leave

    -- After 20 seconds, close the parent dialog.  This
    -- will also close the child dialog.
    if i == 20 then do
      parentDlg~ok
      leave
    end
  end

::requires "ooDialog.cls"

::class 'ExampleChildDlg' subclass UserDialog

::method init
  forward class (super) continue
  self~create(60, 70, 257, 123, "Example - I am the Child Dialog")

::method defineDialog

  self~createPushButton(IDOK, 142, 99, 50, 14, "DEFAULT", "Ok")
  self~createPushButton(IDCANCEL, 197, 99, 50, 14, , "Cancel")


::class 'SimpleParentDlg' subclass UserDialog

::method init
  forward class (super) continue

  self~create(30, 30, 257, 123, "Simple Parent Dialog", "CENTER")

::method defineDialog

  self~createPushButton(IDOK, 142, 99, 50, 14, "DEFAULT", "Ok")
  self~createPushButton(IDCANCEL, 197, 99, 50, 14, , "Cancel")
```

4.5.36. redrawControl

```
>>--redrawControl(--id--+-------------+--)------><
                        +-,--erasebkg-+
```

The redrawControl method redraws the specified dialog control.

Arguments:

The arguments are:

id

The ID of the button

erasebkg

Determines whether (1) or not (0) the background of the drawing area should be erased before redrawing. The default is 0.

4.5.37. redrawRect

```
Form 1:

>>--redrawRect(--+--------+-,--rectangle--+---------+--)----------------------><
```

```
                 +--hwnd--+              +-,-erase-+

Form 2:

>>--redrawRect(--+--------+-,--pt1--,--pt2--+---------+--)--------------------><
                 +--hwnd--+            +-,-erase-+

Form 3:

>>--redrawRect(--+--------+-,--x-,--y-,--x1-,--y1--+---------+--)-------------><
                 +--hwnd--+                   +-,-erase-+

Generic form:

>>--redrawRect(--+--------+-,--rectCoordinates--+---------+--)----------------><
                 +--hwnd--+                +-,-erase-+
```

The *redrawRect* method redraws the specified *bounding* rectangular within the client area of this dialog. Optionally, another window can be specified, and the background of the rectangle can be erased first.

Arguments:

The arguments are:

hwnd [optional]

By default the rectangle to redraw is assumed to belong to the client area of this dialog. Optionally, the window *handle* of some other window can be specified.

rectCoordinates [required]

The coordinates of the rectangle to be redrawn. The coordinates specify the upper left and lower right corners of the rectangle. The corners can be specified using either a *Rect* object, two *Point* objects, or the individual x and y coordinates of each corner.

The coordinates are specified as *client area* coordinates, in pixels.

erase [optional]

A boolean (**.true** or **.false**) that specifies whether the background of the rectangle should be redrawn (**.true**) or not **.false**.) The default is **.false**

Return value:

0 on success, 1 on error.

Remarks:

The rectangle is not really erased, but rather the background is specified to be redrawn also. By default the operating system does not redraw the background.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example shows 2 methods in an application that hides some of its dialog controls, under certain circumstances, and unhides them later:

```
::method hideCustomerControls private

  gb = self~newGroupBox(IDC_GB)
  pb = self~newPushButton(IDC_PB)

  rGB = gb~windowRect
  self~screen2client(rGB)

  rLV = list~windowRect
  self~screen2client(rLV)

  rRect = .Rect~new(rGB~left, rGB~top, rLV~right, rLV~bottom)

  gb~hideFast
  list~hideFast
  self~redrawRect( , rRect, .true)

  return 0

::method showCustomerControls private

  gb = self~newGroupBox(IDC_GB)
  pb = self~newPushButton(IDC_PB)

  rLV = list~windowRect
  self~screen2client(rLV)

  rRect = .Rect~new(rGB~left, rGB~top, rLV~right, rLV~bottom)

  gb~showFast
  list~showFast
  self~redrawRect( , rRect)
```

4.5.38. redrawWindow

```
>>--redrawWindow(--hwnd--)-----------------------><
```

The redrawWindow method redraws a specific dialog.

Arguments:

The only argument is:

hwnd

The handle to the dialog that is to be redrawn.

Return value:

0

Redrawing was successful.

1

Redrawing failed.

4.5.39. redrawWindowRect

```
>>--redrawWindowRect(--+--------+--+------------+--)------------><
                       +--hwnd--+  +-,-erasebkg-+
```

The *redrawWindowRect* method, by default, forces this dialog to completely redraw its *client area*.

Arguments:

The arguments are:

hwnd [optional]

Optionally, some other window can be forced to redraw by specifying the *handle* to the window. All Rexx dialog and dialog controls have the *hwnd* attribute which contains the handle of the *underlying* window. In addition, methods like *getControlHandle* can be used to obtain a window handle. If you omit this argument, the handle of the dialog itself is used.

erasebkg [optional]

Specifies whether the background of the client area should be erased before redrawing. If **`.true`** the background is erased, if **`.false`** the background is not erased. The default is **`.false`**.

Return value:

Returns 0 on success, 1 on error.

Details:

Sets the *.systemErrorCode*.

4.5.40. resizeControl

```
>>--resizeControl(--id--,--width--,--height--+-----------+--)----------------><
                                             +-,-showOpt--+
```

The resizeControl method changes the size of a dialog control.

Arguments:

The arguments are:

id

The ID of the dialog item you want to resize

width, height

The new size in dialog units

showOpt

This argument can be one of the following keywords:

HIDEWINDOW

Hides the control.

SHOWWINDOW

Shows the control.

NOREDRAW

Resizes the control without updating the display. Use the *update* method to manually update the display.

Example:

The following example resizes a dialog control:

```
self~resizeControl(IDC_PB_NEXT, 40, 30, "SHOWWINDOW")
```

4.5.41. scrollBitmapFromTo

```
>>--scrollBitmapFromTo(--id--,--fromX--,--fromY--,--toX--,--toY---------------->


>--+---------+--+---------+--+---------+--+------------+--)--------------------><
   +-,-stepX-+  +-,-stepY-+  +-,-delay-+  +-,-displace-+
```

Scrolls, (moves,) a bitmap from one position to another within a bitmap button.

Arguments:

The arguments are:

id [required]

The resource ID of the bitmap button. May be numeric or *symbolic*.

fromX, fromY [required]

The starting position, in client *coordinates*, where the upper left corner of the bitmap is first drawn. This is specified in pixels.

toX, toY [required]

The ending position, in client *coordinates*, where the upper left corner of the bitmap is drawn last. This is specified in pixels.

stepX, stepY [optional]

The amount, in pixels, that the bitmap is moved for each step. For each move, *stepX* is the amount the bitmap is moved horizontally and *stepY* is the amount the bitmap is vertically. Either or both can be negative. The default for both is 0. However, they can not both be 0, or both omitted.

delay [optional]

The time in milliseconds this method waits after each move before doing the next move. This determines the speed at which the bitmap moves. The default is 0, which means there is no delay between each move of the bitmap.

displace [optional]

If **`.true`** the internal position of the bitmap is updated after each incremental move. *setBitmapPosition* is called after each step to adjust the bitmap position. If the dialog is redrawn, the bitmap is shown at the correct position. The default is **`.true`**.

Return value:

Returns 0 on success and 1 on error.

Remarks:

This method is only for bitmaps that have been assigned to a bitmap button through the *installBitmapbutton* method. If there is no bitmap assigned to the button then 1 is returned.

Scrolling is done by positioning the bitmap and then drawing it. After the bitmap is drawn, it is repositioned using the *stepX* and *stepY* arguments and drawn again. This is repeated until the bitmap reaches the (*toX*, *toY*) ending position.

4.5.42. scrollButton

```
>>--scrollButton(--id--,--xPos--,--yPos--,--lft--,--top--,--rght--,--bttm--)---><
```

Moves the specified rectangle within the button and redraws the uncovered area with the button background color. This method is used to move bitmaps within bitmap buttons.

Arguments:

The arguments are:

id [required]

The resource ID of the button. May be numeric or symbolic.

xPos, yPos [required]

The client area *coordinates* of the new position for the upper left corner of the specified rectangle. The coordinates are specified in pixels.

lft, top, rght, bttm [required]

The client area *coordinates* of the upper left and bottom right corners of the rectangular area to be moved. The coordinates are specified in pixels.

Return value:

Returns 0 on success and 1 on error.

Remarks:

This method is only intended to be used with button controls. It will fail if the resource ID argument specifies a dialog control that is not a button.

The *underlying* Windows dialog must exist to use this method.

Details:

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

4.5.43. scrollInControl

```
>>--scrollInControl(--id-,-text--+--------+--+--------+--+--------+--+-----+--->
                                 +-,-font-+  +-,-size-+  +-,-opts-+  +-,-y-+

>--+--------+--+---------+--+---------+--------------------------------------><
   +-,-step-+  +-,-sleep-+  +-,-color-+
```

Scrolls text in a dialog control using the specified font.

Arguments:

The arguments are:

id [required]

The resource id of the dialog control where the text will be scrolled. May be numeric or *symbolic*.

text [required]

A text string that is displayed and scrolled.

font [optional]

The name of the font used to write the text. If omitted, the System font is used.

size [optional]

The size of the font used to write the text. If omitted, the default is 10.

opts [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. These options control aspects of the font and how the font is written. If this argument is omitted a normal font is used, i.e., not bolded, underlined, italicized, or striked out.

THIN	SEMIBOLD	UNDERLINE
EXTRALIGHT	EXTRABOLD	ITALIC
LIGHT	HEAVY	STRIKEOUT
MEDIUM	BOLD	

THIN

The weight of the font in a range of 0 through 1000 will be 100.

EXTRALIGHT

The weight of the font in a range of 0 through 1000 will be 200.

LIGHT

The weight of the font in a range of 0 through 1000 will be 300.

MEDIUM

The weight of the font in a range of 0 through 1000 will be 500.

SEMIBOLD

The weight of the font in a range of 0 through 1000 will be 600.

BOLD

The weight of the font in a range of 0 through 1000 will be 700.

EXTRABOLD

The weight of the font in a range of 0 through 1000 will be 800.

HEAVY

The weight of the font in a range of 0 through 1000 will be 900.

UNDERLINE

An underline font is used.

ITALIC

An italic font is used.

STRIKEOUT

A strike out font is used.

displaceY

The Y client area *coordinates* of the text. The default is 0. This argument is used to shift the text down from the top edge of the control, if needed.

step

The amount of screen pixels that the text is moved in each iteration. The default is 4.

sleep

The time, in milliseconds, that the program waits after each iteration. This determines the scrolling speed. The default is 10 milliseconds.

color

The color index used for the text. The default is 0, which is black.

Return value:

Returns 0 on success and 1 on error.

Remarks:

The text is scrolled from right to left. Scrolling is done by repeatedly rewriting the text at *step* pixels to the left in the client area of the dialog control, starting at the right edge of the control. When the writing position reaches the left edge of the control, the rewriting ends.

The *font*, *size*, *style*, and *color* arguments specify a font for the text being scrolled. The ooDialog framework requests a font to match this description from the font manager.

If the method is started concurrently, call it a second time with no arguments to stop the scrolling.

Example:

The following example scrolls the string "Hello world!" from left to right within the dialog control window with the symbolic resource id of IDC_PB_SCROLLER. The text is located 2 pixels below the top of the client area, one move is 3 screen pixels, and the delay time after each movement is 15 ms.

```
self~start(scrollInControl, IDC_PB_SCROLLER, "Hello world!", "Arial", 36,   -
                            "BOLD ITALIC", 2, 3, 15, 4)
```

4.5.44. scrollText

```
>>--scrollText(--hwnd-,-text--+--------+--+--------+--+--------+--+-----+------>
                              +-,-font-+  +-,-size-+  +-,-opts-+  +-,-y-+

>--+--------+--+---------+--+---------+----------------------------------------><
   +-,-step-+  +-,-sleep-+  +-,-color-+
```

Scrolls text in the window using the specified font.

Arguments:

The arguments are:

hwnd [required]

The window *handle* where the text will be scrolled.

text [required]

A text string that is displayed and scrolled.

font [optional]

The name of the font used to write the text. If omitted, the System font is used.

size [optional]

The size of the font used to write the text. If omitted, the default is 10.

opts [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. These options control aspects of the font and how the font is written. If this argument is omitted a normal font is used, i.e., not bolded, underlined, italicized, or striked out.

THIN	SEMIBOLD	UNDERLINE
EXTRALIGHT	EXTRABOLD	ITALIC
LIGHT	HEAVY	STRIKEOUT

MEDIUM BOLD

THIN
The weight of the font in a range of 0 through 1000 will be 100.

EXTRALIGHT
The weight of the font in a range of 0 through 1000 will be 200.

LIGHT
The weight of the font in a range of 0 through 1000 will be 300.

MEDIUM
The weight of the font in a range of 0 through 1000 will be 500.

SEMIBOLD
The weight of the font in a range of 0 through 1000 will be 600.

BOLD
The weight of the font in a range of 0 through 1000 will be 700.

EXTRABOLD
The weight of the font in a range of 0 through 1000 will be 800.

HEAVY
The weight of the font in a range of 0 through 1000 will be 900.

UNDERLINE
An underline font is used.

ITALIC
An italic font is used.

STRIKEOUT
A strike out font is used.

displaceY
The Y client area *coordinates* of the text. The default is 0. This argument is used to shift the text down from the top edge of the window, if needed.

step
The amount of screen pixels that the text is moved in each iteration. The default is 4.

sleep
The time, in milliseconds, that the program waits after each iteration. This determines the scrolling speed. The default is 10 milliseconds.

color
The color index used for the text. The default is 0, which is black.

Return value:
Returns 0 on success and 1 on error.

Remarks:
The text is scrolled from right to left. Scrolling is done by repeatedly rewriting the text at *step* pixels to the left in the client area of the window, starting at the right edge of the window. When the writing position reaches the left edge of the window, the rewriting ends.

The *font*, *size*, *style*, and *color* arguments specify a font for the text being scrolled. The ooDialog framework requests a font to match this description from the font manager.

If the method is started concurrently, call it a second time with no arguments to stop the scrolling.

Example:

The following example scrolls the string "Hello world!" from left to right within the control window with the symbolic resource id of IDC_PB_SCROLLER. The text is located 2 pixels below the top of the client area, one move is 3 screen pixels, and the delay time after each movement is 15 ms.

```
hwnd = self~newPushButton(IDC_PB_SCROLLER)~hwnd

self~start(scrollText, hwnd, "Hello world!", "Arial", 36,    -
                       "BOLD ITALIC", 2, 3, 15, 4)
```

4.5.45. setBitmapPosition

```
Form 1:

>>--setBitmapPosition(--id--,--point--)---------><

Form 2:

>>--setBitmapPosition(--id--,--x--,--y--)-------><

Generic form:

>>--setBitmapPosition(--id--,--newPosition--)---><
```

Sets the position of the upper left corner of the bitmap within the *client area* of a bitmap button.

Arguments:

The arguments are:

id [required]

The resource ID of the button. May be numeric or symbolic.

newPosition [required]

The *newPosition* argument(s) specify the (x, y) coordinates of the bitmap. These coordinates can be specified either as a *Point* or as 2 separate whole number arguments, as in Form 2.

Whether specified as a **`point`** object, or using the *x, y* format, both the x and y coordinates are required. If only one argument is used, it must be a **`point`** object. The coordinates are specified as client area coordinates, in pixels.

Return value:

This method returns 0 for success and 1 on error.

Remarks:

Recall that client area coordinates are relative to the (0, 0) point of the client area of a window. Both or either of the coordinates can be negative. A negative x will place the bitmap to the left of the edge of the button. A negative y will place the bitmap above the top of the bitmap.

This method is only for bitmaps that have been assigned to a bitmap button through the *installBitmapbutton* method. If there is no bitmap assigned to the button then 1 is returned.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

```
    p = .Point~new
    if button~getBitmapPosition(IDC_PB_HORSEBITMAP, p) then
        p~x += 4
        p~y += 3
        button~setBitmapPosition(IDC_PB_HORSEBITMAP, p)
    end
```

4.5.46. setControlColor

```
>>--setControlColor(--id--,--+------+--+-------+--+---------+--)---------------><
                             +-,-bk-+  +-,-fg--+  +-,-isClr-+
```

Sets the background color or the text color, or both, for the specified dialog control.

Arguments:

The arguments are:

id [required]

The resource ID of the dialog control whose color is being set. May be numeric or *symbolic*.

bk [optional]

Specifies the background color of the dialog control. This may either be a palette *color number*, or a *COLORREF* number. To use a COLORREF, the *isClr* argument must be true.

fg [optional]

Specifies the foreground color of the dialog control. This may either be a palette *color number*, or a *COLORREF* number. To use a COLORREF, the *isClr* argument must be true. The foreground color is the color that text is written in.

isClr [optional]

Specifies if the *bg* and *fg* arguments are to be interpreted as palette indexes or COLORREF numbers. The default if omitted is false.

Return value:

Returns 0 on success, -1 if a symbolic ID is used for the resource ID and it could not be resolved, or 1 on some other error.

Remarks:

If the *bk* argument is omitted, then ooDialog ensures that the background of the dialog control is painted with the same color as the dialog background.

Both the *bk* and *fg* arguments must be specified in the same manner. They must both be palette indexes or COLORREF numbers.

Earlier versions of ooDialog had a restriction on the number of different dialog controls that could have their control changed from their default color. In ooDialog version 4.2.0 that restriction was lifted and there is no limit to the number of controls having their color set.

Example:

This example changes the background color of a static control to red and the text color to yellow using palette indexes:

```
self~setControlColor(IDC_ST_EDIT_LABEL, 2, 4)
```

4.5.47. setControlFont

```
>>--setControlFont(--id--,--fonthandle--+-----------+--)--------><
                                        +-.-redraw--+
```

The setControlFont method changes the font for a particular dialog control.

The best place to call this method is within *initDialog*. If the font is no longer needed, for instance, when the dialog is closed or another font has been assigned to the dialog control, you should free the font resource by calling *deleteFont*. A good place to do this is the *leaving* method.

Arguments:

The arguments are:

id

The ID of the dialog item.

fonthandle

The handle returned by *createFontEx*.

redraw

0

Do not redraw the control.

1

Redraw the control, which is the default.

Example:

The following example sets a 12-point Arial font for the control:

```
::method initDialog
     .
     .
     .
   hFont = self~createFontEx("Arial",12)
   self~setControlFont(IDC_LB_ADDRESSES, hFont, .false)
```

4.5.48. setControlSysColor

```
>>--setControlSysColor(--id--,--+------+--+-------+--)----------><
                                +-,-bk-+  +-,-fg--+
```

Sets the background color or foreground color, or both, for the specified dialog control using the specified system *color*.

Arguments:

The arguments are:

id [required]

The resource ID of the dialog control whose color is being set. May be numeric or *symbolic*.

bk [optional]

The system color ID for the background color of the dialog control. This can be either the non-negative whole number ID or the keyword ID. IDs can be looked up in the System Color Elements *table*.

fg [optional]

The system color for the foreground color of the dialog control. The foreground color is the color that text is written in. This can be either the non-negative whole number ID or the keyword ID. IDs can be looked up in the System Color Elements *table*.

Return value:

Returns 0 on success, -1 if a symbolic ID is used for the resource ID and it could not be resolved, or 1 on some other error.

Remarks:

If the *bk* argument is omitted, the ooDialog framework ensures that the operating system paints the background using the background color of the dialog.

Earlier versions of ooDialog had a restriction on the number of different dialog controls that could have their control changed from their default color. In ooDialog version 4.2.0 that restriction was lifted and there is no limit to the number of controls having their color set.

Example:

This example changes the background color of a static control to the system background color for a menu:

```
self~setControlSysColor(IDC_ST_STATES, 'MENU')
```

4.5.49. setForegroundWindow

```
>>--setForegroundWindow(--hwnd--)----------------><
```

Brings the specified window to the foreground.

Arguments:

The only argument is:

hwnd [required]

The *handle* to the window that is to become the foreground window.

Return value:

The window handle of the previous foreground window, or 0 if this method failed.

Remarks:

When a window becomes the foreground window the operating system puts the thread that created the specified window into the foreground and activates the window. The window receives the keyboard input, and visual cues are changed to alert the user of the new foreground window. The system also gives a slightly higher priority to the thread that created the foreground window than it does to other threads.

Windows no longer allows a program to arbitrarily change the foreground window. The system restricts which processes can set the foreground window. A process can set the foreground window only if one of the following conditions is true:

- The process is the foreground process.
- The process was started by the foreground process.
- The process received the last input event.
- There is no foreground process.
- No menus are active.

With this change, an application cannot force a window to the foreground while the user is working with another window.

Only top-level windows can become the foreground window. If *hwnd* specifies a dialog control window, its owner dialog becomes the foreground window and the dialog control receives the input focus.

In very rare cases, there might not be a previous foreground window and 0 would be returned. In this case, the **`.systemErrorCode`** will be 0, otherwise the **`.systemErrorCode`** will not be 0.

Details:

Sets the *.systemErrorCode*.

Example:

This example makes another dialog the foreground window and returns the window handle of the current foreground window to the caller, or returns -1 if there was an error:

```
::method switchToReportDlg
  expose reportDlg
  currentHwnd = self~setForeGroundwindow(reportDlg~dlgHandle)
  if currentHwnd == 0 then return -1  -- Report an error
  return currentHwnd
```

4.5.50. setListColumnWidth

```
>>--setListColumnWidth(--id--,--width--)---------><
```

The *setListColumnWidth* method sets the width of all columns in a list box, in dialog units.

The method usually produces *inaccurate* results because it converts the dialog units to pixels using *factorX*. The programmer is advised to use the *setListColumnWidthPx* method instead, which will produce the correct results.

Arguments:

The arguments are:

id [required]

The resource ID of the list box for which you want to set the column width. May be numeric or *symbolic* .

width [required]

The width, in pixels, to set the columns in the list box.

Return value:

Returns -1 if *id* is not correct, otherwise 0.

4.5.51. setListColumnWidthPx

```
>>--setListColumnWidthPx(--id--,--width--)-------><
```

The *setListColumnWidthPx* method sets the width of all columns in a list box, in pixels.

Arguments:

The arguments are:

id [required]

The resource ID of the list box for which you want to set the column width. May be numeric or *symbolic* .

width [required]

The width, in pixels, to set the columns in the list box.

Return value:

Returns -1 if *id* is not correct, otherwise 0.

4.5.52. setListItemHeight

```
>>--setListItemHeight(--id--,--height--)---------><
```

The *setListItemHeight* method sets the height for all items in a list box, in dialog units.

The method usually produces *inaccurate* results because it converts the dialog units to pixels using *factorX*. The programmer is advised to use the *setListItemHeightPx* method instead, which will produce the correct results.

Arguments:

The arguments are:

id [required]

The resource ID of the list box for which you want to set the item height. May be numeric or *symbolic*

height [required]

The height to set the items in the list box to, in dialog units.

Return value:

Returns -1 if *id* is not correct, otherwise 0.

4.5.53. setListItemHeightPx

```
>>--setListItemHeightPx(--id--,--height--)-------><
```

The *setListItemHeightPx* method sets the height for all items in a list box, in pixels.

Arguments:

The arguments are:

id [required]

The resource ID of the list box for which you want to set the item height. May be numeric or *symbolic*

height [required]

The height to set the items in the list box to, in pixels.

Return value:

Returns -1 if *id* is not correct, otherwise 0.

4.5.54. setListWidth

```
>>--setListWidth(--id--,--scrollwidth--)---------><
```

The *setListWidth* method sets the scrollable width of a list box, in dialog units.

The method usually produces *inaccurate* results because it converts the dialog units to pixels using *factorX*. The programmer is advised to use the *setListWidthPx* method instead, which will produce the correct results.

Arguments:

The arguments are:

id [required]

The ID of the list box for which you want to set the scrollable width. May be numeric or *symbolic*.

scrollwidth [required]

The width of the scrollable area of the list box, in dialog units.

Return value:

Returns -1 if *id* is not correct, otherwise 0.

Remarks:

A list box will display a horizontal scrollbar only if it has the *createListBox* style **and** the scrollable width of the list box is set wider than the width of the list box.

When the *underlying* list box is created it does not have any items and the list box sets its scrollable width to its own width. The list box does not update its scrollable width dynamically, so if an item is added to the list box whose width is wider than the list box, no scrollbar will appear. Likewise, for the user to be able to scroll enough to see all items in the list box, the scrollable width must be set to at least the width of the widest item in the list.

4.5.55. setListWidthPx

```
>>--setListWidthPx(--id--,--scrollwidth--)-------><
```

The *setListWidthPx* method sets the scrollable width of a list box, in pixels.

Arguments:

The arguments are:

id [required]
: The ID of the list box for which you want to set the scrollable width. May be numeric or *symbolic*.

scrollwidth [required]
: The width of the scrollable area of the list box, in pixels.

Return value:

Returns -1 if *id* is not correct, otherwise 0.

Remarks:

A list box will display a horizontal scrollbar only if it has the *createListBox* style **and** the scrollable width of the list box is set wider than the width of the list box.

When the *underlying* list box is created it does not have any items and the list box sets its scrollable width to its own width. The list box does not update its scrollable width dynamically, so if an item is added to the list box whose width is wider than the list box, no scrollbar will appear. Likewise, for the user to be able to scroll enough to see all items in the list box, the scrollable width must be set to at least the width of the widest item in the list.

4.5.56. setSBPos

```
>>--setSBPos(--id--,--pos--+----------+--)-------><
                           +-,-redraw-+
```

The *setSBPos* method sets the current position of a *ScrollBar* bar control.

Arguments:

The arguments are:

id
: The resource ID of the scroll bar whose position is to be set. May be numeric or *symbolic*.

pos
: The new position for the scroll bar. It must be within the defined *setSBRange* of the scroll bar.

redraw
: **`.true`** or **`.false`**, specifying whether or not the scroll bar should redraw itself after it is repositioned. The default is **`.true`**.

4.5.57. setSBRange

```
>>--setSBRange(--id--,--min--,--max--,--+--------+--)----------><
                                        +-redraw-+
```

The setSBRange method sets the range of a scroll bar control. It sets the minimum and maximum values.

Protected:

This method is not intended to be used outside of the BaseDialog class.

Arguments:

The arguments are:

id
: The ID of a scroll bar control.

min
: The minimum value.

max
: The maximum value.

redraw
: A flag indicating whether (1) or not (0) the scroll bar should be redrawn. The default is 1.

Example:

The following example allows the scroll bar to take values between 1 and 10:

```
MyDialog~setSBRange(234, 1, 10, 1)
```

4.5.58. setWindowRect

```
Form 1:

>>--setWindowRect(--hwnd--,--rectangle--+---------+--)-------------------------><
                                        +-,-opts--+

Form 2:

>>--setWindowRect(--hwnd--,--pt--,--size--+---------+--)-----------------------><
                                          +-,-opts--+

Form 3:

>>--setWindowRect(--hwnd--,--x-,--y-,--cx-,--cy--+---------+--)----------------><
                                                 +-,-opts--+

Generic form:

>>--setWindowRect(--hwnd--,--rectCoordinates--+---------+--)-------------------><
                                              +-,-opts--+
```

Changes the size and position of the specified window. By specifying either the NOSIZE or NOMOVE options the programmer can only move or only resize the window.

Arguments:

The arguments are:

hwnd [required]
: The *handle* of the window to be resized and / or moved.

rectCoordinates [required]
: The coordinates for the new size and position of the window as a *point/size* rectangle. The coordinates specify the upper left point of the new position, the width of the window and the height of the window. These coordinates can be specified using either a *Rect* object, a *Point* and a *Size* object, or the individual x and y coordinates, width, and height values.

The coordinates are specified as *client area* coordinates, in pixels. All 4 of the coordinates opts [optional]

A list of 0 or more of the following keywords separated by spaces. Case is not significant. If this argument is omitted, the window is made visible, moved to the position specified by the x, y coordinates and resized to the width and height specified by the width and height portion of the point / size rectangle

NOSIZE	HIDEWINDOW	NOREDRAW
NOMOVE	SHOWWINDOW	

NOSIZE

The window will not be resized. The new width and height specified is ignored. The values can be 0, or any other value.

NOMOVE

The position of the window will not be changed. The new x, y position is ignored. Again, the values can be 0, or any other value.

HIDEWINDOW

In addition to any moving or resizing that is done, the window is made invisible.

SHOWWINDOW

In addition to any moving or resizing that is done, the window is made visible.

NOREDRAW

No repainting of any kind is done. The programmer is responsible for invalidating or redrawing any parts of the window and **parent** window that need redrawing. See the *redraw* method.

Return value:

Returns 0 for success, 1 on error. On error, check the **`.systemErrorCode`**.

Remarks:

Note carefully: If a **`Rect`** object is used to specify the point / size rectangle, the semantics are not quite correct. The *right* member of the **`Rect`** must contain the width of the window and the *bottom* member must contain the height of the window.

When the *showOpts* argument contains the NOMOVE keyword, the x, y coordinates are ignored. Likewise, if *showOpts* contains NOSIZE, the width and height portion of the point / size rectangle is ignored.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example resizes an edit control so that it takes up the entire *client area* of the dialog:

```
::method resizeEditControl
  expose editControl isHidden

  hwnd = self~getControlHandle(IDC_MULTILINE)
  rect = self~clientRect

  self~setWindowRect(hWnd, rect)

  if isHidden then do
    editControl~show
    isHidden = .false
  end
```

4.5.59. writeToWindow

```
>>--writeToWindow(-h-,-x-,-y-,-txt-+-------+-+------+-+--------+-+------+-+------+-)--><
                                   +-,-fnt-+ +-,-fs-+ +-,-opts-+ +-,-fg-+ +-,-bk-+
```

The *writeToWindow* method writes text to the specified dialog or dialog control in the given font, style, and color, at the specified position.

Arguments:

The arguments are:

h [required]

The window *handle* of the dialog or dialog control to write the text to.

x, y [required]

The starting position of the text, in pixels. The position coordinates are relative to the window, by default, or the client area of the window, not relative to the screen. Use the CLIENT keyword in the *opts* argument to indicate the starting position is relative to the client area of the window.

txt [required]

The text string to be written.

fnt [optional]

The font name. The default if omitted is SYSTEM.

fs [optional]

The point size of the font. If omitted, the standard size (10) is used.

opts [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. These options control aspects of the font and how the font is written.

OPAQUE	LIGHT	BOLD
TRANSPARENT	MEDIUM	UNDERLINE
CLIENT	SEMIBOLD	ITALIC
THIN	EXTRABOLD	STRIKEOUT
EXTRALIGHT	HEAVY	

OPAQUE

The background of the area the text will occupy is painted with the specified background color, or with white if the background color is omitted, before writing the text. This has the effect of "erasing" whatever is currently drawn in that area. Contrast this with the TRANSPARENT option. OPAQUE is the default.

TRANSPARENT

The background area of the text is left unchanged. (The background color option is ignored if it is used.) This has the effect of writing the text over the top of whatever is currently drawn in the area the text will occupy. Contrast this with the OPAQUE option.

CLIENT
: The position for the text will be relative to the *client area* of the specified window, rather than relative to the window itself.

THIN
: The weight of the font in a range of 0 through 1000 will be 100.

EXTRALIGHT
: The weight of the font in a range of 0 through 1000 will be 200.

LIGHT
: The weight of the font in a range of 0 through 1000 will be 300.

MEDIUM
: The weight of the font in a range of 0 through 1000 will be 500.

SEMIBOLD
: The weight of the font in a range of 0 through 1000 will be 600.

BOLD
: The weight of the font in a range of 0 through 1000 will be 700.

EXTRABOLD
: The weight of the font in a range of 0 through 1000 will be 800.

HEAVY
: The weight of the font in a range of 0 through 1000 will be 900.

UNDERLINE
: An underline font is used.

ITALIC
: An italic font is used.

STRIKEOUT
: A strike out font is used.

fg [optional]
: The color index for the text foreground color. If omitted, the text color is left unchanged.

bk [optional]
: The color index of the background color. If omitted, the background color is left unchanged. The background color is not used in transparent mode.

Return value:

Returns 0 on success and 1 on error.

Remarks:

This method sets **`.systemErrorCode`** to the error code set by the operating system when a failure in one of the Win32 APIs is detected. However, there is one Win32 API, **`SelectObject()`**, that does not set the system error code on failure. It is unlikely that it will fail, but if it does, the ooDialog framework sets **`.systemErrorCode`** to **`156`**, ERROR_SIGNAL_REFUSED.

The text message for error code **`156`** is: *The recipient process has refused the signal.* In this case, the text message is not really related to the failure, it is just used to indicate that the **`SelectObject()`** API failed.

Details:

Sets the *.systemErrorCode*. See the remarks above.

Example:

The following example writes the string *Hello world!* on the surface of a button using a blue 24pt Arial font in bold, italic style. The starting position of the text is at the point (3, 3) relative to the client area of the button:

```
hwnd = self~NewPushButton(IDC_PB_CANVAS)
self~writeToWindow(hwnd, 3, 3, "Hello world!", "Arial", 24, "BOLD ITALIC CLIENT", 4)
```

4.5.60. writeToControl

```
>>--writeToControl(-id-,-x-,-y-,-txt-+-------+-+------+-+--------+-+------+-+------+-)-><
                                     +-,-fnt-+ +-,-fs-+ +-,-opts-+ +-,-fg-+ +-,-bk-+
```

The *writeToControl* method writes text to the specified dialog control in the given font, style, and color, at the specified position.

Arguments:

The arguments are:

id [required]

The resource ID of the dialog control to write the text to. May be numeric or *symbolic*.

x, y [required]

The starting position of the text, in pixels. The position coordinates are relative to the window, by default, or the client area of the window, not relative to the screen. Use the CLIENT keyword in the *opts* argument to indicate the starting position is relative to the client area of the window.

txt [required]

The string to be written.

fnt [optional]

The font name. The default if omitted is SYSTEM.

fs [optional]

The point size of the font. If omitted, the standard size (10) is used.

opts [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. These options control aspects of the font and how the font is written.

OPAQUE	LIGHT	BOLD
TRANSPARENT	MEDIUM	UNDERLINE
CLIENT	SEMIBOLD	ITALIC
THIN	EXTRABOLD	STRIKEOUT
EXTRALIGHT	HEAVY	

OPAQUE

The background of the area the text will occupy is painted with the specified background color, or with white if the background color is omitted, before writing the text. This has the effect of "erasing" whatever is currently drawn in that area. Contrast this with the TRANSPARENT option. OPAQUE is the default.

TRANSPARENT
: The background area of the text is left unchanged. (The background color option is ignored if it is used.) This has the effect of writing the text over the top of whatever is currently drawn in the area the text will occupy. Contrast this with the OPAQUE option.

CLIENT
: The position for the text will be relative to the *client area* of the specified window, rather than relative to the window itself.

THIN
: The weight of the font in a range of 0 through 1000 will be 100.

EXTRALIGHT
: The weight of the font in a range of 0 through 1000 will be 200.

LIGHT
: The weight of the font in a range of 0 through 1000 will be 300.

MEDIUM
: The weight of the font in a range of 0 through 1000 will be 500.

SEMIBOLD
: The weight of the font in a range of 0 through 1000 will be 600.

BOLD
: The weight of the font in a range of 0 through 1000 will be 700.

EXTRABOLD
: The weight of the font in a range of 0 through 1000 will be 800.

HEAVY
: The weight of the font in a range of 0 through 1000 will be 900.

UNDERLINE
: An underline font is used.

ITALIC
: An italic font is used.

STRIKEOUT
: A strike out font is used.

fg [optional]
: The color index for the text foreground color. If omitted, the text color is left unchanged.

bk [optional]
: The color index of the background color. If omitted, the background color is left unchanged. The background color is not used in transparent mode.

Return value:

Returns 0 on success and 1 on error.

Remarks:

This method sets **`.systemErrorCode`** to the error code set by the operating system when a failure in one of the Win32 APIs is detected. However, there is one Win32 API, **`SelectObject()`**, that does not set the system error code on failure. It is unlikely that it will fail, but if it does, the ooDialog framework sets **`.systemErrorCode`** to **`156`**, ERROR_SIGNAL_REFUSED.

The text message for error code **`156`** is: *The recipient process has refused the signal.* In this case, the text message is not really related to the failure, it is just used to indicate that the **`SelectObject()`** API failed.

Details:

Sets the *.systemErrorCode*. See the remarks above.

Example:

The following example writes the string *I am done!* to the client area of a static control. The font of the text will be a blue 18pt Arial font in bold, italic style. The starting position of the text is at the point (5, 5) relative to the client area of the static control.

```
self~writeToControl(IDC_ST_BLANK, 5, 5, "I am done!", "Arial", 18, "BOLD ITALIC CLIENT",
 4)
```

4.6. DynamicDialog Mixin Class

The ooDialog framework allows the Rexx programmer to dynamically define a dialog *template* in her code. This functionality is implemented by the **`DynamicDialog`** class. This class is inherited by the *UserDialog*and its methods are fully documented there.

Historically the ooDialog documentation did not distinguish the difference between the **`UserDialog`** methods and the **`DynamicDialog`** methods. The documentation merely treated the **`DynamicDialog`** methods as methods of the **`UserDialog`**. The **`DynamicDialog`** methods are listed here as a reference with a link to the full documentation in the **`UserDialog`** chapter.

4.6.1. Method Table

The following table lists the methods of the **`DynamicDialog`** class. The links in the table lead to the full documentation for each method, which is in the *UserDialog Class* chapter.

Table 4.8. DynamicDialog Method Reference

Dialog Method	Description
Instance Methods	**Instance Methods**
addIconResource	Adds the file name for an icon resource to the dialog's icon table.
create	Begins the creation of the in-memory Windows dialog template by specifying the general details for the dialog itself.
createBitmapButton	Creates a bitmap push button in the dialog template.
createBlackFrame	Creates a static black frame control in the dialog template.
createBlackRect	Creates a static black rectangle control in the dialog template.
createCenter	Begins the creation of an in-memory dialog template for a dialog that will be centered in the user's screen.
createCheckBox	Creates a check box control in the dialog template.

Dialog Method	Description
createCheckBoxGroup	Creates a group of check box controls in the dialog template using a single string to specify the check box labels.
createCheckBoxStem	Creates a group of check box controls in the dialog template using a stem to specify the check box labels.
createComboBox	Creates a combobox control in the dialog template.
createComboBoxInput	Creates a combobox with a static text control (for the label) in the dialog template.
createDateTimePicker	Creates a date time picker control in the dialog template.
createEdit	Creates an edit control in the dialog template
createEditInput	Creates an edit control and a static text control (for the label) in the dialog template, in one step.
createEditInputGroup	Creates one or more edit controls with labels in the dialog template using a single string to specify the labels.
createEditInputStem	Creates one or more edit controls with labels in the dialog template using a stem to specify the labels.
createEtchedFrame	Creates a static etched frame control in the dialog template.
createEtchedHorizontal	Creates a static control in the dialog template that is an etched horizontal line.
createEtchedVertical	Creates a static frame control in the dialog template that is a single etched vertical line.
createGrayFrame	Creates a static gray frame control in the dialog template.
createGrayRect	Creates a static gray rectangle control in the dialog template.
createGroupbox	Creates a group box in the dialog template.
createListBox	Creates a list box control in the dialog template.
createListView	Creates a list-view control in the dialog template.
createMonthCalendar	Creates a month calendar control in the dialog template.
createOkCancelRightBottom	Creates an Ok and Cancel push buttons in the right bottom corner of the dialog template.
createOkCancelRightTop	Creates an Ok and Cancel push buttons in the right top corner of the dialog template.
createOkCancelLeftBottom	Creates an Ok and Cancel push buttons in the left bottom corner of the dialog template.
createOkCancelLeftTop	Creates an Ok and Cancel push buttons in the left top corner of the dialog template.
createPasswordEdit	Creates an edit control with the PASSWORD style set in the dialog template.
createProgressBar	Creates a progress bar control in the dialog template.
createPushButton	Creates a push button in the dialog template.
createPushButtonGroup	Creates any number of push buttons in the dialog template at one time.
createRadioButton	Creates a radio button control in the dialog template.
createRadioButtonGroup	Creates a group of radio buttons with labels in the dialog template using a single string to specify the labels.

Dialog Method	Description
createRadioButtonStem	Creates a group of radio buttons with labels in the dialog template using a stem to specify the labels.
createScrollBar	Creates a scroll bar control in the dialog template.
createStatic	Creates a static control in the dialog template.
createStaticFrame	Creates one of the static frame controls in the dialog template.
createStaticImage	Creates a static image control in the dialog template.
createStaticText	Creates a static text control in the dialog template.
createTab	Creates a tab control in the dialog template.
createToolBar	Creates a toolbar control in the dialog template.
createTrackbar	Creates a trackbar control in the dialog template.
createTreeView	Creates a tree view control in the dialog template
createUpDown	Creates an up-down control in the dialog template.
createWhiteFrame	Creates a white static frame control in the dialog template.
createWhiteRect	Creates a white static rectangle control in the dialog template.
defineDialog	The programmer uses this method to dynamically create the dialog controls in the dialog template by using the appropriate create ... methods.
load	Creates the in-memory dialog template by reading the dialog template from a resource script file.
loadFrame	Starts the in-memory Windows dialog template by reading a dialog template from a resource script file.
loadItems	Creates the dialog controls in the dialog template, and finishes the dialog template, by reading a dialog template from a resource script file.

4.6.2. addIconResource

```
DynamicDialog::addIconResource


>>--addIconResource(--id--,--fileName--)---------><
```

4.6.3. create

```
DynamicDialog::create


>>--create(--x--,--y--,--cx--,--cy--,--title-+-----------+--+------------+----->
                                             +-,-options-+  +-,-dlgClass-+

>----+------------+--+------------+--+------------+--)------------------------><
     +-,-fontName-+  +-,-fontSize-+  +-,-expected-+
```

4.6.4. createBitmapButton

```
DynamicDialog::createBitmapButton
```

```
>>--createBitmapButton(-id-,-x-,-y-+------+-+------+-+----------+-+--------+--->
                                   +-,-cx-+ +-,-cy-+ +-,-style--+ +-,-text-+

>--+------------+-,-normal-+------------+--+-------------+--+-------------+--)--><
   +-,-methName-+          +-,-focused--+  +-,-selected--+  +-,-disabled--+
```

4.6.5. createBlackFrame

```
DynamicDialog::createBlackFrame


>>--createBlackFrame(--+------+--x-,-y-,-cx-,-cy--+-----------+-)--------------><
                       +--id--+                   +-,--style--+
```

4.6.6. createBlackRect

```
DynamicDialog::createBlackRect


>>--createBlackRect(--+------+--x-,-y-,-cx-,-cy--+-----------+-)---------------><
                      +--id--+                   +-,--style--+
```

4.6.7. createCenter

```
DynamicDialog::createCenter


>>--createCenter(--cx--,--cy--,--title-+-----------+--+------------+----------->
                                       +-,-options-+  +-,-dlgClass-+

>----+------------+--+------------+--+------------+--)------------------------><
     +-,-fontName-+  +-,-fontSize-+  +-,-expected-+
```

4.6.8. createCheckBox

```
DynamicDialog::createCheckBox


>>--createCheckBox(-id-,--x-,--y--+------+--+------+--+---------+--+---------+->
                                  +-,-cx-+  +-,-cy-+  +-,-style-+  +-,-label-+

>----+-----------------+--+--------------+--)----><
     +-,-attributeName-+  +-,-connectOpt-+
```

4.6.9. createCheckBoxGroup

```
DynamicDialog::createCheckBoxGroup


                                             +-----+
                                             V     |
>>--createCheckBoxGroup(id,-x-,-y-+------+-,---txt-+--+---------+-+----------+-)--><
                                  +-,-cx-+            +-,-style-+ +-,-idStat-+
```

4.6.10. createCheckBoxStem

```
DynamicDialog::createCheckBoxStem


>>--createCheckBoxStem(id,-x-,-y-+------+-,-txt.-+-------+-+---------+-+----------+-)-><
                                 +-,-cx-+        +-,-max-+ +-,-style-+ +-,-idStat-+
```

4.6.11. createComboBox

```
DynamicDialog::createComboBox


>>--createComboBox(--id--,--x--,--y--,--cx--,--cy--+---------+--+------------+--)-><
                                                   +-,-style-+  +-,-attrName-+
```

4.6.12. createComboBoxInput

```
DynamicDialog::createComboBoxInput


>>--createComboBoxInput(-id-,-x-,-y-+-------+-,-cx2-+---------+-,--text------->
                                 +-,-cx1-+       +-,-count-+

>---+---------+---+----------+--+----------+--)------------------------------><
    +-,-style-+   +-,-idStat-+  +-,-attName-+
```

4.6.13. createDateTimePicker

```
DynamicDialog::createDateTimePicker


>>--createDateTimePicker(-id--x-,--y-,--cx-,--cy-+---------+--+------------+--)-><
                                                 +-,-style-+  +-,-attrName-+
```

4.6.14. createEdit

```
DynamicDialog::createEdit


>>--createEdit(-id-,-x-,-y-,-cx--+------+--+---------+--+------------+--)------><
                                 +-,-cy-+  +-,-style-+  +-,-attrName-+
```

4.6.15. createEditInput

```
DynamicDialog::createEditInput


>>--createEditInput(-id-,-x-,-y-+-------+-,-cx2-,-+------+-,-text-+----------+->
                                +-,-cx1-+         +-,-cy-+         +-,-style-+

>--+-----------+-+--------+-)-------------------------------------------------><
   +-,-idStatic-+ +-,-attr-+
```

4.6.16. createEditInputGroup

```
DynamicDialog::createEditInputGroup


                                                        +---+
                                                        V   |
>>--createEditInputGroup(-id-,-x-,-y-+-------+-,-cx2-,--txt-+-+------+-+-------+-)-><
                                     +-,-cx1-+                 +-,-s-+ +-,-id2-+
```

4.6.17. createEditInputStem

```
DynamicDialog::createEditInputStem


>>--createEditInputStem(-id-,-x-,-y-+-------+-,-cx2-,-text.--------------------->
                                    +-,-cx1-+

>--+---------+-+----------+-)--------------------------------------------------><
   +-,-style-+ +-,-idStat-+
```

4.6.18. createEtchedFrame

```
DynamicDialog::createEtchedFrame


>>--createEtchedFrame(--+-------+--x-,-y-,-cx-,-cy--+-----------+-)------------><
                        +--id-,-+                   +-,--style--+
```

4.6.19. createEtchedHorizontal

```
DynamicDialog::createEtchedHorizontal


>>--createEtchedHorizontal(--+-------+--x-,-y-,-cx-,-cy--+-----------+-)-------><
                             +--id-,-+                   +-,--style--+
```

4.6.20. createEtchedVertical

```
DynamicDialog::createEtchedVertical


>>--createEtchedVertical(--+-------+--x-,-y-,-cx-,-cy--+-----------+-)---------><
                           +--id-,-+                   +-,--style--+
```

4.6.21. createGrayFrame

```
DynamicDialog::createGrayFrame


>>--createGrayFrame(--+------+--x-,-y-,-cx-,-cy--+-----------+-)---------------><
                      +--id--+                   +-,--style--+
```

4.6.22. createGrayRect

```
DynamicDialog::createGrayRect


>>--createGrayRect(--+------+--x-,-y-,-cx-,-cy--+-----------+-)----------------><
                     +--id--+                   +-,--style--+
```

4.6.23. createGroupbox

```
DynamicDialog::createGroupbox


>>--createGroupbox(-+------+-,-x-,-y-,-cx-,-cy--+---------+-+--------+-)-------><
                    +--id--+                    +-,-style-+ +-,-text-+
```

4.6.24. createListBox

```
DynamicDialog::createListBox


>>--createListBox(--id-,-x-,-y-,-cx-,-cy--+---------+-+------------+-)---------><
                                          +-,-style-+ +-,-attrName-+
```

4.6.25. createListView

```
DynamicDialog::createListView


>>--createListView(-id-,--x-,--y-,--cx-,--cy-+---------+--+------------+--)----><
                                             +-,-style-+  +-,-attrName-+
```

4.6.26. createMonthCalendar

```
DynamicDialog::createMonthCalendar


>>--createMonthCalendar(-id--x-,--y-,--cx-,--cy-+---------+--+------------+--)-><
                                                +-,-style-+  +-,-attrName-+
```

4.6.27. createOkCancelLeftBottom

```
DynamicDialog::createOkCancelLeftBottom


>>--createOkCancelLeftBottom---------------------><
```

4.6.28. createOkCancelLeftTop

```
DynamicDialog::createOkCancelLeftTop
```

```
>>--createOkCancelLeftTop------------------------><
```

4.6.29. createOkCancelRightBottom

```
DynamicDialog::createOkCancelRightBottom


>>--createOkCancelRightBottom--------------------><
```

4.6.30. createOkCancelRightTop

```
DynamicDialog::createOkCancelRightTop


>>--createOkCancelRightTop-----------------------><
```

4.6.31. createPasswordEdit

```
DynamicDialog::createPasswordEdit


>>--createPasswordEdit(-id-,-x-,-y-,-cx--+------+--+---------+-+--------+--)---><
                                         +-,-cy-+  +-,-style-+ +-,-attr-+
```

4.6.32. createProgressBar

```
DynamicDialog::createProgressBar


>>--createProgressBar(--id--,--x--,--y--,--cx--,--cy-+-----------+--)----------><
                                                     +--,-style--+
```

4.6.33. createPushButton

```
DynamicDialog::createPushButton


>>--createPushButton(-id-,-x-,-y-,-cx-,-cy-,-+---------+-+-------+-+-------+-)-><
                                             +-,-style-+ +-,-txt-+ +-,-mth-+
```

4.6.34. createPushButtonGroup

```
DynamicDialog::createPushButtonGroup


                                                 +----------------+
                                                 V                |
>>--createPushButtonGroup(-x-,-y-+------+-+------+-,-txt- -id- -m-+--+-----+-+-----+-)-><
                                 +-,-cx-+ +-,-cy-+                   +-,-r-+ +-,-o-+
```

4.6.35. createRadioButton

```
DynamicDialog::createRadioButton


>>--createRadioButton(-id-,-x-,-y-+------+--+------+--+---------+-------------->
                                  +-,-cx-+  +-,-cy-+  +-,-style-+

>--+--------+--+--------+--)---------------------------------------------------><
   +-,-text-+  +-,-attr-+
```

4.6.36. createRadioButtonGroup

```
DynamicDialog::createRadioButtonGroup



                                                 +-----+
                                                 V     |
>>--createRadioButtonGroup(id1,-x-,-y-+------+-,---txt-+--+---------+-+--------+-)--><
                                      +-,-cx-+            +-,-style-+ +-,-idSt-+
```

4.6.37. createRadioButtonStem

```
DynamicDialog::createRadioButtonStem


>>--createRadioButtonStem(-id-,-x-,-y--+------+-,-text.--+-------+------------->
                                       +-,-cx-+          +-,-max-+

>--+---------+--+----------+-)-------------------------------------------------><
   +-,-style-+  +-,-idStat-+
```

4.6.38. createScrollBar

```
DynamicDialog::create createScrollBar


>>--createScrollBar(--id--,--x--,--y--,--cx--,--cy--+----------+--)------------><
                                                    +-,-style--+
```

4.6.39. createStatic

```
DynamicDialog::createStatic


>>--createStatic(--+----+--,-x-,-y-,-cx-,-cy--+----------+--+---------+--)-----><
                   +-id-+                     +-,-style--+  +-,-text--+
```

4.6.40. createStaticFrame

```
DynamicDialog::createStaticFrame
```

```
>>--createStaticFrame(--id--,-x--,-y--,-cx--,-cy--+----------+--)--------------><
                                                  +-,-style--+
```

4.6.41. createStaticImage

```
DynamicDialog::createStaticImage


>>--createStaticImage(--id--,-x--,-y--,-cx--,-cy--+----------+--)--------------><
                                                  +-,-style--+
```

4.6.42. createStaticText

```
DynamicDialog::createStaticText


>>--createStaticText(-+------+--x-,--y-+------+-+------+-+----------+-+--------+-)-><
                      +-id-,-+         +-,-cx-+ +-,-cy-+ +-,-style--+ +-,-text-+
```

4.6.43. createWhiteFrame

```
DynamicDialog::createWhiteFrame


>>--createWhiteFrame(--+------+--x-,-y-,-cx-,-cy--+-----------+-)--------------><
                       +--id--+                   +-,--style--+
```

4.6.44. createWhiteRect

```
DynamicDialog::createWhiteRect


>>--createWhiteRect(--+------+--x-,-y-,-cx-,-cy--+-----------+-)---------------><
                      +--id--+                   +-,--style--+
```

4.6.45. createTab

```
DynamicDialog::createTab


>>--createTab(-id-,--x-,--y-,--cx-,--cy-+---------+--+-----------------+--)----><
                                        +-,-style-+  +-,-attributeName-+
```

4.6.46. createToolBar

```
DynamicDialog::createToolBar


>>--createToolBar(-id-,--x-,--y-,--cx-,--cy-+---------+--+-----------------+--)-><
                                            +-,-style-+  +-,-attributeName-+
```

4.6.47. createTrackBar

```
DynamicDialog::createTrackbar


>>--createTrackBar(-id-,--x-,--y-,--cx-,--cy-+---------+--+------------+--)----><
                                             +-,-style-+  +-,-attrName-+
```

4.6.48. createTreeView

```
DynamicDialog::createTreeView


>>--createTreeView(-id-,--x-,--y-,--cx-,--cy-+---------+--+------------+--)----><
                                             +-,-style-+  +-,-attrName-+
```

4.6.49. createUpDown

```
DynamicDialog::createUpDown


>>--createUpDown(-id-,--x-,--y-,--cx-,--cy-+---------+--+-----------------+--)-><
                                           +-,-style-+  +-,-attributeName-+
```

4.6.50. defineDialog

```
DynamicDialog::defineDialog


>>--defineDialog---------------------------------><
```

4.6.51. load

```
DynamicDialog::load


>>--load(--rcFile--+-------+--+----------+--+-------------+--)-----------------><
                   +-,-id--+  +--,-opts--+  +-,--expected-+
```

4.6.52. loadFrame

```
DynamicDialog::loadFrame


>>--loadFrame(--rcFile--+-------+--+----------+--+-------------+--)------------><
                        +-,-id--+  +--,-opts--+  +-,--expected-+
```

4.6.53. loadItems

```
DynamicDialog::loadItems
```

```
>>--loadItems(--rcFile--+-------+--+----------+--)----------------------------><
                        +-,-id--+  +--,-opts--+
```

4.7. EventNotification Mixin Class

The event notification class is a *Mixin* class that contains methods that connect the notification of a Windows *event* sent to the *underlying* dialog with a method in the Rexx dialog object. Except as noted below, the Rexx programmer adds the method to her dialog subclass. Once the event notification is connected, each time the event occurs, the connected method is invoked. The programmer codes the method to take the appropriate action for the event.

In the Windows user interface, as the user interacts with the system, events are generated that specify what the action of the user was. Mouse clicks, keyboard presses, moving or sizing windows, all generate events.

Some simple examples of how this works:

- Push buttons notify their parent dialog of a number of events. The programmer connects one of these events to a method in his Rexx dialog. Typically the CLICKED event is connected. Then, the connected method is called each time the button is pressed (clicked.)
- List boxes, multiple-select list boxes, and combo boxes can be connected to a method that is called each time a line in the list box or combo box is selected.
- For a scroll bar, the programmer can specify different methods that are called depending on the user action. The user can click on the arrow buttons, drag the thumb, or use the page down / page up keys. Each of these events can be connected to a method in the Rexx dialog.

Whenever a dialog object is instantiated, the ooDialog framework *automatically* makes several event connections. For all other events the Rexx programmer, usually, needs to specifically connect the event to his dialog, through one of the event notification methods, to be able to respond to the event. In a *UserDialog*, some of the methods that create dialog controls have an option that automatically connects an event to a dialog method. But, for most events, say the resize or move events, the connection needs to be made explicitly. Indeed, except for trivial dialogs, most of the programming is deciding which events to be notified of and taking action upon receiving the notification.

4.7.1. Connecting Event Handlers

In general, events should be connected before the dialog is shown on the screen, or immediately after the dialog is created. There is no reason why the programmer can not place the invocation where ever it makes the most sense. Connecting the events in the *initDialog*() method is usually sufficient. In a **`UserDialog`**, the *defineDialog*() method also makes a good place for the connect event methods. If the programmer is overriding the *init*() method, the connect event methods can be placed there. In this case, do not invoke the connect event method until **after** the superclass has been *initialized*.

4.7.2. Coding Event Handlers

The methods in a Rexx dialog object connected to Windows event notifications are commonly called event handlers because the method handles the event. In order to properly code an event handler, the Rexx programmer needs to understand somewhat the underlying mechanism of event notification. This is particularly important in ooDialog 4.2.0 and later with its ability to *directly* invoke the event handling method from the window *message* processing loop.

When the operating system sends an event notification to the underlying dialog, the operating system waits for a reply to the notification in the message processing loop. This is significant because while the operating system is waiting, in general, other messages can not be processed by the window. If an application does not reply in a timely manner to the notification messages, it can cause the application to appear hung. With today's fast processors there are plenty of CPU cycles for an event handler to process and reply to event notifications. But, if an application were to do something like sleep for 10 seconds in an event handler, the user would think the application was hung.

Prior to ooDialog 4.2.0, the ooDialog framework could not directly invoke the event handling method in the Rexx dialog object. To invoke the method, the framework used a cumbersome process involving queues and interpret. In the message processing loop, the framework essentially queued up the method to be invoked *at some later time* and replied immediately to the notification. Because of this, the event handling method could not reply to the event notification, and in fact any return was simply lost. This meant much of the capability of a Windows dialog was unavailable. It also meant it did not matter if the method returned in a timely manner.

With ooDialog 4.2.0 and later, the event handling method in the Rexx dialog object is invoked by the interpreter directly from the message processing loop in the underlying Windows dialog. This allows the Rexx programmer to *return* a value to the notification, to *synchronize* the handling of notifications, or to *veto* an event. Remember, with this direct invocation of the event handling methods, the interpreter is waiting in the message processing loop for the return from the method. Therefore, the programmer needs to return from the method in a reasonable amount of time. Do not be overly worried about this, the interpreter processes a large number of statements in a very short time. There is adequate time to process and return a value in the event handler.

However, having the interpreter waiting for the return of an event handling method could conceivably be a problem for some few existing programs. Since, in older ooDialog versions, the return from an event handler method was meaningless, some programmers may not have returned an actual value from the method, and / or they may not have returned from the method in a timely manner. The programmer may not have realized this was a mistake.

In addition, there are a large number of event notifications in Windows where the operating system ignores the reply. Therefore, the ooDialog framework uses three different ways to invoke the event handling method. This is controlled by the *willReply* argument that is common to all event connection methods.

The first, preferred, method is for the interpreter to invoke the method directly, wait for the return, and then use the return value to reply to the Windows event notification. The second is to return 0 immediately to the operaring system, and invoke the event handling method to run concurrently as a separate activity. The third is to invoke the event handler directly, wait for the reply from the event handler, but ignore the returned value.

Event handling methods must be public

Event handling methods must be public, they *can not* be private. The dialog object has no way of knowing an event has happened. Some object, some thing, outside of the dialog object, knows of the event. That other object must notify the dialog that the event has happened. In this case the other object is the operating system.

Private methods of an object can not be invoked from outside of the object. If an event handling method is private, there is no way for the operating system to invoke that method to notify the dialog that the event has happened.

4.7.3. Coding Event Handler Examples

The following code snippets and examples expand on the discussion of how to code event handling methods.

Best practice would be to always code event handlers as if they are expected to return a value to the operating system's event notification message. The considerations would be that the method does not block and that a value is returned to the operating system in a reasonable amount of time. In general, the event handler should be unguarded to preclude the possibility that some guarded method in the dialog object is executing at the time the event notification is generated.

Below is a code snippet for an application that displays the current date and time, with a push button that refreshes the display. Note these points about the code snippet. Since the *connectButtonEvent* method is the replacement for the *deprecated* *connectButtonNotify* it is in essence an existing connection method, using the original arguments. Therefore, by default, the method is invoked from the message processing loop to run concurrently as a separate activity. Because of this, technically, it does not have to be unguarded and does not have to return a value. Nevertheless, the example is the *correct* way to code the event handler.

```
::method initDialog
  ...
  self~connectButtonEvent(IDC_PB_REFRESH, "CLICKED", updateTime)
  ...

::method updateTime unguarded
  use arg info, hwnd

  now = .DateTime~new
  self~refreshDisplay(now)
  return 0
```

Consider this snippet from a similar program, but where the push button starts a process that calculates the grains of sand in the universe. Since it is not expected that the calculation will finish in a reasonable amount of time, an early reply is used to return a value in a timely manner. The handler also disables the push button so that the user can not start a second calculation until the first calculation finishes.

```
::method initDialog
  ...
  self~connectButtonEvent(IDC_PB_CALC_SAND, "CLICKED", onCalculateGrains)
  ...

::method onCalculateGrains unguarded
  use arg info, hwnd

  self~newPushButton(IDC_PB_CALC_SAND)~disable
  reply 0

  number = self~calculateSandInUniverse
  self~refreshDisplay(number)
  self~newPushButton(IDC_PB_CALC_SAND)~enable
  return
```

The following examples are all for events where having the interpreter wait for the returned value from the event handler is the best way to handle the notifications.

Returning Values

The *MonthCalendar* control has the *GETDAYSTATE* event that is sent to request information on how certain days are to be shown. The programmer can customize the calendar by returning a set of days that should be displayed in bold. For instance, in a business application paid holidays could be displayed in bold.

Since the *connectMonthCalendarEvent* method did not exist in ooDialog 4.1.0 and the return value for this event controls what the operating system does, the *willReply* argument is forced to true by the ooDialog framework. That is, the interpreter invokes the event handler directly, waits for the reply value, and returns that value to the operating system., The programmer can not change this behavior. If the programmer does not wish to return a meaningful value from the event handler, the event should not be connected.

```
::method initDialog
  expose calendar

  calendar = self~newMonthCalendar(IDC_MC_HOLIDAYS)

  -- Connect the GETDAYSTATE event.
  self~connectMonthCalendarEvent(IDC_MC_HOLIDAYS, "GETDAYSTATE", onGetDayState)

  -- Restrict the calendar so that it only displays the year 2011.
  start = .DateTime~fromStandardDate("20110101")
  end = .DateTime~fromStandardDate("20111231")

  ...

::method onGetDayState unguarded
  expose calendar
  use arg startDate, count

  -- Create the array to hold the .DayState objects.
  dayStates = .array~new(count)

  month = startDate~month
  if month == 12 then month = 0

  do i = 1 to count
    j = month + i - 1

    select
      when j ==  1 then dayStates[i] = .DayState~new(17)
      when j ==  2 then dayStates[i] = .DayState~new(21)
      when j ==  3 then dayStates[i] = .DayState~new
      when j ==  4 then dayStates[i] = .DayState~new
      when j ==  5 then dayStates[i] = .DayState~new(30)
      when j ==  6 then dayStates[i] = .DayState~new
      when j ==  7 then dayStates[i] = .DayState~new(4)
      when j ==  8 then dayStates[i] = .DayState~new
      when j ==  9 then dayStates[i] = .DayState~new(5)
      when j == 10 then dayStates[i] = .DayState~new
      when j == 11 then dayStates[i] = .DayState~new(24, 25)
      when j == 12 then dayStates[i] = .DayState~new(23, 30)
      otherwise dayStates[i] = .DayState~new()
    end
  end

  buffer = .DayStates~makeDayStateBuffer(dayStates)
  return buffer
```

Event Synchronization

The *DateTimePicker* control has the DROPDOWN event notification that is sent when the drop down calendar is shown. This gives the programmer a chance to customize the month calendar that is shown. Since the month calendar is not shown until the event handling method returns, replying directly to the notification allows the programmer to completely finish the customizations before the month calendar appears on the screen.

```
::method initDialog

  self~connectDateTimePickerEvent(IDC_DTP, "DROPDOWN", onDropDown)
  ...


::method onDropDown unguarded
  use arg idFrom, hwndFrom

  dt = self~newDateTimePicker(IDC_DTP);
  monthCal = dt~getMonthCal
  monthCal~setFirstDayOfWeek(3)
  monthCal~addStyle("NOTODAY")

  return 0
```

In the above example, the *connectDateTimePickerEvent* method did not exist in ooDialog 4.1.0. Therefore, by default the interpreter invokes the event handler directly, waits for a reply, and expects a returned value. However, the operating system ignores the actual value of the return. The programmer can change the default behavior if he wants. If there is no reason to have the interpreter wait until the event handler finishes, then the programmer could use the optional *willReply* argument. The **`.false`** value tells the ooDialog framework to invoke the event handler to run concurrently as a separate activity.

```
::method initDialog

  self~connectDateTimePickerEvent(IDC_DTP, "DROPDOWN", onDropDown, .false)
  ...


::method onDropDown unguarded
  expose userDidSeeCalendar
  use arg idFrom, hwndFrom

  userDidSeeCalendar = .true
```

However, if the programmer is going to customize the calendar, having the event handler run concurrently is likely to have the calendar shown before the customization is finished. So, it makes much more sense to at least use the *sync* option in this way:

```
::method initDialog

  self~connectDateTimePickerEvent(IDC_DTP, "DROPDOWN", onDropDown, 'SYNC')
  ...


::method onDropDown unguarded
  use arg idFrom, hwndFrom

  dt = self~newDateTimePicker(IDC_DTP);
  monthCal = dt~getMonthCal
  monthCal~setFirstDayOfWeek(3)
  monthCal~addStyle("NOTODAY")
```

Veto Events

The *Tab* control has the SELCHANGING event. It is sent when the user selects a different tab, and is sent **before** the selected tab is changed. The programmer can *veto* the change to a new tab by returning **`.false`** to the event notification, or allow the change by returning **`.true`**. One reason for preventing the change might be that the user had entered invalid data in the current tab page.

Note: Since the *connectTabEvent* method is the replacement for the *ovvDeprecated connectTabNotify* method, it essentially is a method that existed in ooDialog 4.1.0. When the same arguments are used as existed in 4.1.0, the default for *willReply* is false. The event handler is invoked to run concurrently as a separate activity and the interpreter does not wait for a return value. With this behavior, it is impossible for the programmer to veto the change to a new tab. Therefore the programmer has to use the optional *willReply* argument to change the default behavior. Specifying **`.true`** causes the ooDialog framework to invoke the *onTabChanging* method directly, wait for the return value, and pass that value to the operating system.

```
::method defineDialog

  self~connectTabEvent(IDC_TAB, SELCHANGING, onTabChanging, .true)

::method onTabChanging unguarded
  expose tabContent
  use arg idFrom, hwndFrom

  index = self~newTab(idFrom)~selectedIndex + 1
  dlg = tabContent[index]

  if dlg~allowChange then return .true
  else return .false
```

In the above example, since the *allowChange* method is returning **`.true`** or **`.false`** the event handler could have been coded this way:

```
::method onTabChanging unguarded

  ...

  return dlg~allowChange
```

The example used:

```
  if dlg~allowChange then return .true
  else return .false
```

to emphasize that the event handler needs to return **`.true`** or **`.false`**.

Lengthy Processing

As previously noted, the Rexx programmer does not need to be overly worried about taking too much time to return a value from an event handler. In most every case there is plenty of time to process the event and return a value. However, sometimes the programmer may want to, or need to, do some lengthy processing in the event handler. For these cases, the programmer needs to figure out how to return a value from the event handler and also do the needed processing. This is really a *concurrency* problem, not an event handling problem.

The two common ways to solve this problem would be to use an *early reply* or to start a *second activity* running to do the processing. Using the *onTabChanging* event handler above, here are two typical ways to handle the event when some lengthy processing is involved. As a hypothetical, say that when the user changes to a new tab, the program needs to gather up all the data on the validated page and write it to disk, or send it somewhere. One approach would be to validate the page, do an early reply, and then finish up the processing:

```
::method defineDialog
```

```
  self~connectTabEvent(IDC_TAB, SELCHANGING, onTabChanging, .true)

::method onTabChanging unguarded
  expose tabContent
  use arg idFrom, hwndFrom

  index = self~newTab(idFrom)~selectedIndex + 1
  dlg = tabContent[index]

  if \ dlg~allowChange then return .false  -- Tab will not be changed

  reply .true  -- Tab is changed for the user.

  -- Now gather up the data entered on the page we just switched from.
  data = dlg~getUserDataDirectory
  self~writeDataToFile(data)
```

Another approach could be to start a new activity which will run concurrently. It might be coded this way:

```
::method onTabChanging unguarded
  expose tabContent
  use arg idFrom, hwndFrom

  index = self~newTab(idFrom)~selectedIndex + 1
  dlg = tabContent[index]

  if dlg~allowChange then do
    dp = .DataProcesser~new
    dp~start("processDlgData", dlg)
    return .true
  end
  else do
    return .false
  end
```

4.7.4. Common *willReply* argument

It is the intent that every event connection method have an optional *willReply* argument to give the programmer some control of how the interpreter invokes the event handler. For each event connection method, 3 values are accepted for the *willReply* argument:

.true

When *willReply* is true, the interpreter waits for the return value from the event handler and enforces that the method has returned a value. That is, the event handling method must return some value. That value is returned to the operating system.

.false

When *willReply* is false, the interpreter immediately returns 0 to the operating system. It then invokes the event handling method to run concurrently as a separate activity. The return from the event handling method, if any, is ignored.

sync

When *willReply* is the keyword *sync*, the interpreter waits for the return from the event handler, but it discards the value returned. It does not enforce that a value is actually returned.

The default value if the argument is omitted varies depending on the event connection method. In general, the default for event connections that existed prior to ooDialog 4.2.0 is false and for event connections that were added in ooDialog 4.2.0 and afterwards is true.

4.7.5. Method Table

The following table list the methods of the **`EventNotification`** class:

Table 4.9. EventNotification Methods

Method	Description
addUserMsg	Connects an operating system window message with a method in the Rexx dialog object.
connectActivate	Connects the window activation event to a method in the Rexx dialog.
connectAllSBEvents	Connects all event notifications from a scroll bar control to a single method in the Rexx dialog.
connectButtonEvent	Connects an event notification from a button control to a method in the Rexx Dialog.
connectComboBoxEvent	Connects an event notification from a combo box to a method in the Rexx dialog.
connectCommandEvents	Connects a command event notification from a dialog control to a method in the Rexx dialog.
connectDateTimePickerEvent	Connects an event notification form a date time picker to a method in the Rexx dialog.
connectDraw	Connects the draw item event notification to a method in the Rexx dialog.
connectEachSBEvent	Connects each specified event notification from a scroll bar to a separate method in the Rexx dialog.
connectEditEvent	Connects an event notification from an edit control to a method in the Rexx dialog.
connectFKeyPress	Connects a F Key key press (a F key is typed) with a method in the Rexx dialog.
connectHelp	Connects the Windows Help event to a method in the Rexx dialog.
connectKeyPress	Connects a key press (a key is typed) with a method in the Rexx dialog.
connectListBoxEvent	Connects an event notification from a list box control to a method in the Rexx dialog.
connectListViewEvent	Connects an event notification from a list-view control to a method in the Rexx dialog.
connectMonthCalendarEvent	Connects an event notification from a month calendar to a method in the Rexx dialog.
connectMove	Connects the move event notification to a method in the Rexx dialog.
connectNotifyEvent	Connects a generic event notification from a dialog control to a method in the Rexx dialog.
connectPosChanged	Connects the position has changed event notification to a method in the Rexx dialog.

Method	Description
connectResize	Connects the size event notification to a method in the Rexx dialog
connectResizing	Connects the sizing event notification to a method in the Rexx dialog
connectScrollBarEvent	Connects an event notification from a scroll bar control to a method in the Rexx dialog.
connectSizeMoveEnded	Connects the size / move ended event notification to a method in the Rexx dialog object.
connectStaticEvent	Connects an event notification from a static control to a method in the Rexx dialog.
connectTabEvent	Connects an event notification from a tab control to a method in the Rexx dialog.
connectToolBarEvent	Connects an event notification from a toolbar control to a method in the Rexx dialog.
connectToolTipEvent	Connects an event notification from a ToolTip control to a method in the Rexx dialog.
connectTrackBarEvent	Connects an event notification from a track bar control to a method in the Rexx dialog.
connectTreeViewEvent	Connects an event notification from a tree view control to a method in the Rexx dialog.
connectUpDownEvent	Connects an event notification from an UpDown control to a method in the Rexx dialog.
defListDragHandler	Default implementation of a drag and drop handler for a list-view control.
defTreeDragHandler	Default implementation of a drag and drop handler for a tree view control.
disconnectKeyPress	Disconnects a method that was previously connected to key press event.
hasKeyPressConnection	Queries if a connection to a key press event already exists.

4.7.6. addUserMsg

```
>>--addUserMsg(mth-,-winMsg-+------+-+------+-+------+-+------+-+------+-+----------+)--><
                            +-,-f1-+ +-,-wP-+ +-,-f2-+ +-,-lP-+ +-,-f3-+ +-,-wlRply-+
```

The *addUserMsg* method connects a Windows window *message*, with specific parameters, sent to the *underlying* dialog with a method in the Rexx dialog.

Note

This method is designed to be used by ooDialog programmers who are familiar with the Windows API. It's use will require access to the Windows *documentation* at a minimum, and will likely also require access to the Windows header files in the Windows platform *SDK*.

Each added user message and each connected event generates a message entry in an ooDialog table. Internally, ooDialog examines every window message, and its parameters, that the Windows dialog receives. If the window message and its parameters match an entry in the message table, ooDialog invokes the method specified in the message table. For *addUserMsg*, this is the method specified by the *mth* argument.

Arguments:

All arguments, except for *mth*, *rxMsg*, and *wlRply*, represent whole numbers in their hexadecimal format. For convenience, the programmer can either use the numeric value for these argument, or the numbers can be expressed in conventional *hexadecimal* format. It is important to realize that since bitwise *and* is used with *winMsg*, *wP*, and *lP* and their respective filters, it is the hexadecimal format of the whole numbers that is relevant.

The arguments are:

mth [required]
: The method in the Rexx dialog to invoke when, or if, the window message and its parameters sent to the underlying dialog match a message entry in the message table.

winMsg [required]
: The numeric value of the window message to match.

f1 [optional]
: A filter to apply to *winMsg*. The filter is bitwise *anded* with *winMsg* and if the result equals the actual window messge sent, it is considered a match. If omitted a filter of 0xFFFFFFFF is used.

wP [optional]
: The numeric value of the *WPARAM* parameter in the window message to match. This defaults to 0.

f2 [optional]
: A filter to apply to the *wP* argument. The filter is bitwise *anded* with *wP* and if the result equals the actual numeric value of the wParam sent, it is considered a match. If omitted a filter of 0 is used.

lP [optional]
: The numeric value of the *LPARAM* parameter in the window message to match. This defaults to 0.

f3 [optional]
: A filter to apply to the *lP* argument. The filter is bitwise *anded* with *lP* and if the result equals the numeric value of the actual lParam sent, it is considered a match. If omitted a filter of 0 is used. Note that WM_NOTIFY messages are treated slightly different, see the remarks.

wlRply [optional]

The *wlRply* argument controls how the interpreter invokes the event handler for the connected event The default if *willReply* is omitted is `.false`.

Return value:

The return value is 0 for success and 1 for failure.

Remarks:

This method is not intended for every ooDialog programmer. In particular, it requires some understanding of window messages, the message parameters, and bitwise *and*. If the ooDialog programmer does not know what a bitwise *and* is, that is a good indication that this method is not for them.

Incorrect use can crash your program.

If the *wlRply* is set to .true, an incorrect return from the connected event handler can crash your application.

The ooRexx interpreter strives to prevent the ooRexx programmer from shooting himself in the foot. However, the assumption is that if the programmer uses the *addUserMsg* method, despite the warnings, that he knows what he is doing. When *wlRply* is true, the value returned from the event handler is passed on to the operating system as is. It is possible, although unlikely, that an incorrect value could cause the program to crash. For instance, if the operating system is expecting a pointer to memory and the numeric value passed back from the event handler is not pointing to valid memory, the program will crash.

Details for all window messages and their parameters are available in the Windows *documentation*. The numeric value of the message IDs (and possibly the message parameters) can be looked up in the Windows platform *SDK*.

In all cases, it is preferable to use specific connect event methods provided by ooDialog. Such as *connectEditEvent*, *connectResize*, etc.. *addUserMsg* is provided for those cases where the specific connect event method the programmer needs is not present in ooDialog.

WM_NOTIFY is special cased.

The WM_NOTIFY window message is handled as a special case. The LPARAM argument sent with the WM_NOTIFY message has a number of pieces of information embedded within it. One of those pieces is the numeric notification code for the message. This is the important piece that the ooDialog framework uses to filter the messages, Therefore, if the *winMsg* argument is WM_NOTIFY, the *lP* argument should be the notification code for the desired message and the filter should be *0xffffffff*.

A quick example: Windows Vista adds a new style to buttons called a *split button*. Split buttons send a new notification message, BCN_DROPDOWN, when the user clicks on the down arrow of the button. Before ooDialog added support for the split button, a programmer could have used it in their ooDialog programs by creating a dialog template in a resource only DLL with a split button and using *addUserMsg* to capture the drop down notification. Looking up the numeric values for this notification, the programmer would have found that WM_NOTIFY == '0x004E' and that BCN_DROPDOWN == '0xfffffb20'. Assuming the ID for the split button is 1004. The programmer would code *addUserMsg* like this:

```
  addUserMsg('onDropDown', '0x004E', '0xFFFFFFFF', 1004, '0xFFFFFFFF', '0xfffffb20',
 '0xFFFFFFFF', .true)
```

Note that the arguments the event handling method receives are dependent on the window message being connected. There are 3 types of event handlers:

WM_NOTIFY handlers:
The event *handler* for WM_NOTIFY messages receives 4 arguments.

WM_COMMAND handlers:
The event *handler* for WM_COMMAND messages receives 5 arguments.

All Others
The event *handler* for all other messages receives 2 arguments.

Note that some of the arguments sent to the event handlers will be the numeric value of the WPARAM and LPARAM parameters for the window message. If either WPARAM or LPARAM are pointers, the numeric value will not do the ooDialog programmer any good. There is no way in ooDialog Rexx code to access whatever the pointer points to.

The exception to this would be if ooDialog already had a connect event method for the specific window message. In this case the ooDialog framework accesses the memory and sends the correct, appropriate arguments to the event handler. In which case there is no need to use *addUserMsg*.

Filters can be used to ensure that only the specific message desired is connected to the Rexx method, or to match several messages and connect them to the same Rexx method.

Take the window messages WM_KEYDOWN (0x0100) and WM_KEYUP (0x0101.) When the following arguments are used:

```
winMsg == '0x0100'
f1     == '0xFFFF'
```

```
  addUserMsg('onKeyDown', winMsg, f1, 0, 0, 0, 0)
```

only the WM_KEYDOWN message will match and *onKeyDown* is only invoked for that message.

However, when the arguments are:

```
  winMsg == '0x0100'
  f1     == '0xFFFE'
  addUserMsg('onKey', winMsg, f1, 0, 0, 0, 0)
```

both the WM_KEYDOWN and WM_KEYUP messages will match and the *onKey* method will be invoked for either window message.

In a similar fashion, both the LBN_SETFOCUS (0x4) and the LBN_KILLFOCUS (0x5) notifications are sent to a list box using the WM_COMMAND (0x0111) message. For the WM_COMMAND message, the WPARAM parameter contains the control ID in the low word of the parameter and the notification code in the high word of the parameter.

Say the list box control has an ID of 256 (0xFF,) when the arguments are:

```
  winMsg == '0x0111'
  f1     == '0xFFFF'

  wP     == '0x000400FF'
  f2     == '0xFFFFFFFF'
  addUserMsg('onSetFocus', winMsg, f1, wP, f2, 0, 0)
```

only the LBN_SETFOCUS notification sent to the list box with ID of 256 will invoke the *onSetFocus* method.

However, in the following case:

```
  winMsg == '0x0111'
  f1     == '0xFFFF'

  wP     == '0x000400FF'
  f2     == '0xFFFEFFFF'
  addUserMsg('onFocus', winMsg, f1, wP, f2, 0, 0)
```

both the LBN_SETFOCUS and LBN_KILLFOCUS notifications to the listbox will invoke the *onFocus* method.

Details

Raises syntax errors when incorrect usage is detected.

Example:

The WM_CANCELMODE message is sent to cancel certain modes, such as mouse capture. This example shows how an ooDialog program that captures the mouse for some reason could add an event handler for the WM_CANCELMODE message and release the mouse if the program had currently captured it. This is a hypothetical situation intended to show how to use the *addUserMsg* method and not necessarily of any practical value.

```
::method init
    expose mouseIsCaptured
    ...

    WM_CANCELMODE = '0x001F'
    ret = self~addUserMsg('onCancelMode', WM_CANCELMODE, '0xFFFF', 0, 0, 0, 0)
    if ret == 1 then do
        -- Do some error handling stuff.
        -- But, really this is unlikely to happen.
    end

    mouseIsCaptured = .false


::method doCaptureMouse private
    expose mouseIsCaptured

    self~captureMouse
    mouseIsCaptured = .true

    ...  -- more code


::method onCancelMode unguarded
    expose mouseIsCaptured
    use arg wParam, lParam

    -- For WM_CANCELMODE wParam and lParam have no meaning.

    if mouseIsCaptured then do
        self~releaseMouseCapture
        mouseIsCaptured = .false

        ...  -- maybe some more code
    end

    return 0
```

4.7.6.1. addUserMsg Other Message Event Handler

The event handler connected through the *addUserMsg* is invoked when ooDialog internally matches a message sent to the *underlying* Windows dialog using the arguments specified to the *addUserMsg* method. The *other message* event handler is invoked when the window message matched is not WM_NOTIFY or WM_COMMAND.

The *willReply* argument in the *addUserMsg* method determines how the event handler needs to respond to the notification.

```
::method onEvent unguarded
  use arg wParam, lParam

  return 0
```

Arguments:

This event handling method receives two arguments.

wParam

The numeric value of the*WPARAM* parameter sent with the window message that was connected through *addUserMsg*. This numeric value of WPARAM may or may not be of use to the programmer depending on the specific window message.

lParam

The numeric value of the *LPARAM* parameter sent with the window message that was connected through *addUserMsg*. Again, this numeric value of LPARAM may or may not be of use to the programmer depending on the specific window message.

Return:

If the *willReply* argument for the *addUserMsg* method was true, then the event handler *must* return a numeric value. This value is passed on unchanged to the operating system. Otherwise the return, if any, from the event handler is ignored.

Example

The WM_QUERYOPEN message is sent to a window when it is minimized. If the window returns true, it is restored. If it returns false, the window remains minimized. In the following example, if the user minimizes the dialog, when the user goes to restore the dialog, it will only be restored, randomly, half of the time:

```
::method initDialog
    ...

    WM_QUERYOPEN = '0x0013'
    self~addUserMsg(onQueryOpen, WM_QUERYOPEN, '0xFFFFFFFF', 0, 0, 0, 0, .true)
    ...

::method onQueryOpen  unguarded
    use arg wParam, lParam

    n = random(0, 200)

    if n // 2 == 0 then return .true
    else return .false
```

4.7.6.2. addUserMsg WM_COMMAND Message Event Handler

The event handler connected through the *addUserMsg* is invoked when ooDialog internally matches a message sent to the *underlying* Windows dialog using the arguments specified to the *addUserMsg* method. The *WM_COMMAND message* event handler is invoked when the window message matched is WM_COMMAND.

The *willReply* argument in the *addUserMsg* method determines how the event handler needs to respond to the notification.

```
::method onWmCommand unguarded
  use arg wParam, lParam, id, notifyCode, controlObj

  return 0
```

Arguments:

This event handling method receives five arguments. The *wParam* and *lParam* arguments have to be retained for backwards compatibility, but the last 3 arguments are really what are useful.

wParam

The numeric value of the*WPARAM* parameter sent with the window message that was connected through *addUserMsg*. This numeric value will contain the resource ID of the control sending the message and the numeric code of the event that caused the notification to be sent. However, the ooDialog framework extracts these values and sends them as the *id* and *notifyCode* arguments.

lParam

The numeric value of the *LPARAM* parameter sent with the window message that was connected through *addUserMsg*. If a dialog control sent the notification, then the LPARAM value is always the window handle of the control sending the notification. If a menu item or accelerator key sent the notification, then this value will always be zero.

id

The numeric resource id of the object sending the notification. Typically this will be a dialog control, however it could be a menu item or an accelerator key. See the remarks section.

notifyCode

The numeric notification code of the event that caused the notification to be sent. If the notification was sent by a menu item this will always be 0. If sent by an accelerator key, it will always be 1. When the notification is sent by a dialog control, it will be a notification code defined by the control. Note that notification codes do not have unique values. For example, the list box LBN_SETFOCUS notification code has the value of 4, as does the button's BN_DISABLED code.

controlObj

The Rexx dialog control object that represents the underlying dialog control that sent the notification, if a dialog control sent the notification. When a menu item or an accelerator key sent the notification, this will be the **.nil** object.

Return:

If the *willReply* argument for the *addUserMsg* method was true, then the event handler *must* return a numeric value. This value is passed on unchanged to the operating system. Otherwise the return, if any, from the event handler is ignored.

Remarks:

The following table can be used to determine if the *id* argument is a dialog control ID, a menu item ID, or an accelerator key ID:.

Table 4.10. WM_COMMAND Event Handler Arguments

Source	notifyCode	ID	Control Object
Menu	0	Menu item ID	The **.nil** object
Accelerator	1	Accelerator key ID	The **.nil** object
Dialog Control	Control defined code	Resource ID of the control	Rexx dialog control

Example

Button controls send a BN_CLICK notification when the user clicks on a button. Typically radio buttons are created with the BS_AUTORADIOBUTTON style, and the operating system handles setting the checked or unchecked appearance of all the radio buttons in a group. This example shows how to use the *addUserMsg* method to connect the notification to a Rexx method and how to code the event handler to properly set the checked / unchecked appearance of the 2 radio buttons in the group. BN_CLICK notifications are sent using the WM_COMMAND message:

```
::method initDialog
    expose rb1 rb2 BN_CLICK
    ...

    rb1 = self~newRadioButton(IDC_RADIO1)
    rb2 = self~newRadioButton(IDC_RADIO2)

    WM_COMMAND = '0x0111'
    BN_CLICK   = 0
    id         =  rb1~id
    wParam     = .DlgUtil~makeWParam(id, BN_CLICK)
    self~addUserMsg(onRbClick, WM_COMMAND, '0xFFFFFFFF', wParam, '0xFFFFFFFF', -
                    0, 0, .true)

    id     =  rb2~id
    wParam = .DlgUtil~makeWParam(id, BN_CLICK)

    self~addUserMsg(onRbClick, WM_COMMAND, '0xFFFFFFFF', wParam, '0xFFFFFFFF', -
                    0, 0, .true)

    ...

::method onRbClick unguarded
    expose rb1 rb2 BN_CLICK
    use arg wParam, lParam, id, code, rbObj

    if code == BN_CLICK then do
      if id == rb1~id then do
        rb1~check
        rb2~uncheck
      end
      else if id == rb2~id then do
        rb2~check
        rb1~uncheck
      end
    end

  return 0
```

4.7.6.3. addUserMsg WM_NOTIFY Message Event Handler

The event handler connected through the *addUserMsg* is invoked when ooDialog internally matches a message sent to the *underlying* Windows dialog using the arguments specified to the *addUserMsg* method. The *notify message* event handler is invoked when the window message matched is WM_NOTIFY.

The *willReply* argument in the *addUserMsg* method determines how the event handler needs to respond to the notification.

```
::method onWmNotify unguarded
  use arg idFrom, hwndFrom, notifyCode, rexxControl

  return 0
```

Arguments:

This event handling method receives four arguments.

idFrom
: The numeric resource ID of the dialog control sending the notification.

hwndFrom
: The window handle of the dialog control sending the notification.

notifyCode
: The numeric notification code for the event that caused the dialog control to send the notification.

rexxControl
: The Rexx dialog control object that is sending the notification.

Return:

If the *willReply* argument for the *addUserMsg* method was true, then the event handler *must* return a numeric value. This value is passed on unchanged to the operating system. Otherwise the return, if any, from the event handler is ignored.

Example

```
::method initDialog
    ...
    WM_NOTIFY    = '0x004E'
    BCN_DROPDOWN = '0xfffffb20'
    self~addUserMsg('onDropDown', WM_NOTIFY, '0xFFFFFFFF', 2, '0xFFFFFFFF', -
                    BCN_DROPDOWN, '0xFFFFFFFF', .true)
    ...

::method onDropDown unguarded
  use arg idFrom, hwndFrom, notifyCode, rexxControl

  say 'Got split button drop down'
  say 'idFrom:     ' idFrom
  say 'hwndFrom:   ' hwndFrom
  say 'notifyCode: ' notifyCode
  say 'rexxControl:' rexxControl

  return 0

/*  Output would be, note that the window handle will vary:

Got split button drop down
idFrom:      2
hwndFrom:    0x00000000002F0448
notifyCode:  4294966048
rexxControl: a Button

*/
```

4.7.7. connectActivate

```
>>--connectActivate(--+--------------+--+-------------+)-------><
                      +--methodName--+  +-,-willReply--+
```

Connects an *activate* event notification sent to the underlying dialog with a method in the Rexx dialog. This event notification is sent to both the window being activated and the window being deactivated

Arguments:

The arguments are:

methodName [optional]

The name of the method that is invoked each time the dialog gains or loses the activation. The method name can not be the empty string. When this argument is omitted the name defaults to *onActivate*.

willReply [optional]

The *willReply* argument controls how the interpreter invokes the event handler for the connected event. **Note:** All event connection methods accept the *willReply* argument. However, for the *connectActivate* method, the value of the argument is forced to true. That is, event handling method must always return a value to the operating system.

Return value:

0

No error.

1

An (internal) error prevented the message from being connected to a method.

Remarks:

The *active* window is the *top-level* window that the user is currently working with. The activate notification is only sent to top-level windows. The activate notification is always sent in pairs, one notification to the window losing the activation and one to the window gaining the activation. The arguments to the event handler for the notification allow the programmer to determine if the window is gaining or losing the activation.

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

The interpreter invokes the event handler directly and waits in the window *message* processing loop for the return from the event handler. Connecting the activate event requires that the programmer reply to the event from the event handler in a timely manner.

Details:

Syntax errors are raised when incorrect usage is detected.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if any activate events happen.

The underlying dialog receives the WM_ACTIVATE message as the notification for this event.

4.7.7.1. Activate Event Handler

The event handler for the ACTIVATE event is invoked when the dialog window is either losing or gaining the active window status.

The programmer must return a value from the event handler and the interpreter waits for the return value from the event handler.

```
::method onActivate unguarded
  use arg status, hwnd, hFocus, isMinimized

  return .false
```

Arguments:

The event handling method receives four arguments:

status

A keyword that indicates if the dialog is gaining or losing the activation. The keyword will be exactly one of the following:

ACTIVE	CLICKACTIVE	INACTIVE

ACTIVE

The dialog is gaining the activation through some means other than the user clicking the mouse on the window. For example the user may select the window through the ALT-Tab mechanism.

CLICKACTIVE
: The dialog is gaining the activation through a mouse click.

INACTIVE
: The dialog is losing the activation.

hwnd
: The window *handle* of the window being activated or deactivated, depending on the *status* argument. If the keyword is INACTIVE, then this is the handle of the window gaining the activation. If the keyword is ACTIVE or CLICKACTIVE, it is the handle of the window losing the activation.

  Note that this argument may be 0, indicating the operating system did not pass a window handle with the notification.

hFocus
: The window handle of the dialog control with the current focus when the dialog is being deactivated. When the dialog is being activated *hFocus* will be 0.

isMinimized
: Specifies the minimized state of the window being activated or deactivated. *isMinimized* will be **`.true`** if the window is minimized, otherwise **`.false`**.

Return:

The event handler for this notification must return **`.true`** or **`.false`**. A return of true indicates that the notification has been processed and a return of false indicates that the notification has not been processed. When the notification has not been processed the interpreter passes the notification on to the operating system for its default processing.

The default processing done by the operating system includes things like restoring the focus to the dialog control that had the focus when the dialog was deactivated, highlighting the text in an edit control if that control has the focus, etc.. Under most circumstances, the programmer should return **`.false`** to allow the dialog manager to perform this default processing. If not, the programmer should take care of these details or the dialog may not behave as the user expects.

Example

The following example comes from code ooDialog uses internally to handle a problem with a *ListView* control when it is used in a *Tab* control. When the dialog is being inactivated the handle of the focused control is saved. When it is being activated, the handle of the last focused control is passed on the to *ControlDialog* that contains the list-view for processing.

```
::method onActivate unguarded
  expose listViewPageDialog lastFocused
  use arg flag, hwnd, hFocused, isMinimized

  reply .false

  if flag == 'INACTIVE' then lastFocused = hFocused
  else listViewPageDialog~updateListView(lastFocused)
```

4.7.8. connectButtonEvent

```
>>--connectButtonEvent(--id--,--event--+-----------+--+--------------+--)------><
                                       +-,--mName--+  +-,-willReply--+
```

Connects a method in the Rexx dialog to the Windows *event* notification from a *Button* control. The *connectButtonEvent* method is used for all types of buttons (push button, radio button, or check boxes.)

Arguments:

The arguments are:

id [required]

The resource ID of the button control this connection applies to. This can be a symbolic ID or the numeric value of the ID.

event [required]

A keyword specifying the event to be connected with a method. This can be exactly one of the following, case is not significant:

CLICKED

The button has been clicked.

DBLCLK

The button has been double-clicked.

DISABLE

The button has been disabled.

DROPDOWN

Requires Windows version **Vista** or later.

The user has clicked on the down arrow in a *split* button.

GOTFOCUS

The button got the input focus.

LOSTFOCUS

The button lost the input focus.

HILITE

The button has been selected.

HOTITEM

Requires Common Control *Library* version **6.0** or later.

Notifies the dialog that the mouse has moved over the button, or that the mouse is leaving the area over the button.

PAINT

The button is to be repainted. This notification is only sent for owner-drawn buttons.

UNHILITE

The highlighting is to be removed (lost selection).

mName [optional]

The name of the method to invoke whenever the specified notification is received from the button control. Provide a method with a matching name. If you omit this argument, a method name is generated automatically. The name consists of the event keyword preceded by **on**. For instance: **`onGotFocus`**. The method name can not be the empty string.

willReply [optional]
: The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.false`**.

Return value:

The return codes are:

0
: No error detected.

-1
: The resource ID could not be resolved or the event argument is incorrect.

1
: The message was not connected correctly. From ooDialog 4.2.0 and on, it is not likely the return value will be 1.

Remarks:

In order to receive the GOTFOCUS, LOSTFOCUS, and DBLCLK event notifications, the button control has to have the NOTIFY (BS_NOTIFY) style. For user defined dialogs use the NOTIFY style keyword in the *create* method when the button is defined. For dialogs created from a compiled resource or a resource script file use the BS_NOTIFY style for the button resource. The other event notifications are always sent and it is not necessary to add the NOTIFY style for those events.

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

Details:

Syntax errors are raised when incorrect usage is detected.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if any of the button events occur.

In Windows itself, some notifications are sent to the parent dialog using the WM_COMMAND message and others are sent using the WM_NOTIFY message.

Example:

The following example displays a message whenever the OK button is selected:

```
::class MyDlgClass subclass UserDialog

::method init
  self~init:super(...)
  self~connectButtonEvent("OK", "HILITE")

::method onHilite unguarded
  say "The OK button has been selected"
  return 0
```

4.7.8.1. DropDown Event Handler

The event handler for the DROPDOWN event is invoked when the user clicks on the down arrow in a split button. Typically, the programmer uses this notification to display a context menu beneath the split button

The *willReply* argument in the *connectButtonEvent* method determines how the event handler needs to respond to the notification.

```
::method onDropDown unguarded
  use arg id, buttonRect, rxButton

  return 0
```

Arguments:

The event handling method receives 3 arguments:

id

The numeric resource id of the button sending the notification.

buttonRect

A *Rect* object that specifies the size and postion of the button sending the notification. Use this rectangle to properly position a context menu.

rxButton

The Rexx button object that sent the notification.

Return:

The operating system ignores the return from this notification. 0 is a good value to return.

4.7.8.2. General Button Event Handler

The event handler for most of the button events receives the same arguments and is coded in the same fashion. When the handler is invoked is fairly clear from the name of the event. The CLICKED event handler is invoked when the button is clicked, etc..

The *willReply* argument in the *connectButtonEvent* method determines how the event handler needs to respond to the notification.

```
::method onButtonEvent unguarded
  use arg info, hwnd, id, notifyCode, buttonObj

  return 0
```

Arguments:

The event handling method receives 5 arguments. The first and second arguments need to be retained for backwards compatibility, but the last 3 arguments are really what is needed:

info

A numeric value that contains info about the event. The low order word contains the resource ID of the button sending the event. The high order word contains the button event code. However, the ooDialog framework now extracts these values for you and sends them as the third and fourth arguments.

hwnd

The window handle of the button that sent the notification.

id

The numeric resource id of the button sending the notification.

notifyCode
: The numeric notification code of the event that caused the notification to be sent. Each dialog control has its own specific notification codes.

controlObj
: The Rexx button control object that represents the underlying dialog control that sent the notification. It is possible this will be the **`.nil`** object if some error happened, but this is very unlikely.

Return:

The operating system ignores the return from this notification. 0 makes a good return value.

Example

The following example uses one event handler for all the push buttons in the application. It uses the *id* argument to determine which button was pushed:

```
::method onButtonEvent unguarded
  use arg info, handle, id, notifyCode, pushButton

  select
    when id = .constDir[IDC_PB_SLEEP] then do
      reply 0
      pushButton~disable
      self~putToSleep
      pushButton~enable
      return
    end

  when id = .constDir[IDC_PB_LOAD] then do
    reply 0
    pushButton~disable
    self~loadNext
    pushButton~enable
    return
  end

  when id = .constDir[IDC_PB_REPORT] then do
    self~emailStatus
  end

    otherwise nop
  end

  ...

  return 0
```

4.7.8.3. HotItem Event Handler

The event handler for the HOTITEM event is invoked when the user hovers the mouse over a button and when the user moves the mouse off of the area of a button.

The *willReply* argument in the *connectButtonEvent* method determines how the event handler needs to respond to the notification.

```
::method onHotItem unguarded
  use arg id, entering, rxButton

  return 0
```

Arguments:

The event handling method receives 3 arguments:

id

The numeric resource id of the button sending the notification.

entering

This argument is always true or false. If true, the user has started hovering the mouse over the button. If false, the user was hovering the mouse and has now moved the mouse so it is no longer over the button.

rxButton

The Rexx button object that represents the button sending the notification.

Return:

The operating system ignores the value of the return. 0 makes a good value to return.

Example

The following example is from a program that has 2 buttons with the HOTITEM event connected to the same event handler. Each time the event handler is invoked it prints out some information about the event:

```
::method onHover unguarded
  use arg id, entering, rxButton

  if entering then msg = 'mouse is entering'
  else msg = 'mouse is leaving'

  say 'onHover() button with id:' id msg
  ...
  return 0

/* Output might be:

  onHover() button with id: 1044 is entering
  onHover() button with id: 1044 is leaving
  onHover() button with id: 1001 is entering
  onHover() button with id: 1001 is leaving
*/
```

4.7.9. connectComboBoxEvent

```
>>--connectComboBoxEvent(--id--,--event--+----------+--+--------------+--)-----><
                                         +-,--mName-+  +-,-willReply--+
```

Connects a method in the Rexx dialog to the Windows *event* notification from a *ComboBox* control.

Arguments:

The arguments are:

id

The *resource ID* of the combo box for which a notification is to be connected to a method.

event

Exactly one of the following key words, case is not significant, that specified the event to be connected with a method:

CHANGE
: The text in the edit control has been altered. This notification is sent after Windows updated the screen.

CLOSEUP
: The list of the combo box has been closed.

DBLCLK
: An entry in the combo box list has been selected with a double click.

DROPDOWN
: The list of the combo box is about to be made visible.

ERRSPACE
: An out-of-memory problem has occurred.

GOTFOCUS
: The combo box got the input focus.

LOSTFOCUS
: The combo box lost the input focus.

SELCHANGE
: Another entry in the combo box list has been selected.

SELENDCANCEL
: After the selection of another entry, another control or dialog was selected, which canceled the selection of the entry.

SELENDOK
: The list was closed after another entry was selected.

UPDATE
: The text in the edit control has been altered. This notification is sent before Windows updates the screen.

mName
: The name of the method that is to be invoked whenever the specified notification is received from the combo control. Provide a method with a matching name. If you omit this argument, a method name is generated automatically. The name consists of the event keyword preceded by **`on`**. For instance: **`onSelEndOk`**. The method name can not be the empty string.

willReply [optional]
: The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.false`**.

Return value:

The return codes are:

-1
: The resource ID could not be resolved or the event argument is incorrect.

0
: No errors were detected.

1
: The message was not connected correctly.

Remarks:

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

Example:

The following example invokes method PlaySong whenever the list of the combo box with the resource ID of IDC_CB_PROFESSIONS is about to be made visible. In this case IDC_CB_PROFESSIONS is a *symbolic* ID that has been added to the *constDir* directory of the MyDlgClass in another part of the program:

```
::class MyDlgClass subclass UserDialog

::method initDialog
  self~connectComboBoxEvent("IDC_CB_PROFESSIONS", "DROPDOWN", "PlaySong")
```

4.7.9.1. General ComboBox Event Handler

The event handler for all of the combo box events receives the same arguments and is coded in the same fashion. When the handler is invoked is fairly clear from the name of the event. The DROPDOWN event handler is invoked when the drop down arrow is pushed, etc..

The *willReply* argument in the *connectComboBoxEvent* method determines how the event handler needs to respond to the notification.

```
::method onComboBoxEvent unguarded
  use arg info, hwnd, id, notifyCode, comboBox

  return 0
```

Arguments:

The event handling method receives 5 arguments. The first and second arguments need to be retained for backwards compatibility, but only the last 3 arguments are really needed:

info
: A numeric value that contains info about the event. The low order word contains the resource ID of the combo box sending the event. The high order word contains the combo box event code. The *loWord* and *hiWord* methods of the *DlgUtil* class can be used to extract these values. However, the ooDialog framework now extracts those values for you and sends them as the third and fourth arguments.

hwnd
: The window handle of the combo box that sent the notification.

id
: The numeric resource id of the combo box sending the notification.

notifyCode
: The numeric notification code of the event that caused the notification to be sent. Each dialog control has its own specific notification codes.

controlObj
: The Rexx combo box control object that represents the underlying dialog control that sent the notification. It is possible this will be the **`.nil`** object if some error happened, but this is very unlikely.

Return:

The operating system ignores the return from this notification. 0 makes a good return value.

Example

The following example connects the CLOSEUP event of a combo box. The event handler checks if the item selected was Apple, and if it was, the selection is changed to Cherry:

```
::method init

  ,,,

  self~connectComboBoxEvent(IDC_CB_DROPDOWNLIST, CLOSEUP, onCbCloseUp)

::method onCbCloseUp unguarded
  use arg info, hwnd, id, notifyCode, comboBox

  if comboBox~selected == 'Apple' then comboBox~select('Cherry')

  return 0
```

4.7.10. connectCommandEvents

```
>>--connectCommandEvents(--id--,--methodName--+-------------+--)-------------><
                                              +-,-willReply--+
```

The *connectCommandEvents* method connects a Rexx dialog method to the command *event* notifications sent by a Windows dialog control to its parent dialog.

Arguments:

id [required]
: The resource ID of the dialog control, may be symbolic or numeric.

methodName [required]
: The name of the method to be invoked in the Rexx dialog object each time a command event occurs in the dialog control. The method name can not be the empty string.

willReply [optional]
: The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.false`**.

Return value:

-1
: The specified symbolic ID could not be resolved.

0
: No error.

1
: An error. Most likely, either the message table is full, or the interpreter is out of usable memory.

Remarks:

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

The number of different notification codes and the meanings of the notifications are dependent on the type of dialog control specified. Therefore, it is more advisable to use the specific connectXXXEvent() method for the control. Such as the *connectListBoxEvent*() method.

Details:

Syntax errors are raised when incorrect usage is detected.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if any command events happen.

In Windows itself, command events are sent to the parent dialog using the WM_COMMAND message.

Example:

This example connects list box command event notifications to the *onEvent* method of the dialog. Note that the notification code 1 corresponds to the list box SELCHANGE event, 4 is the SETFOCUS, and 5 is KILLFOCUS events.

```
::method initDialog
  ...
  self~connectCommandEvents(IDC_LB_FILES, onEvent)


::method onEvent unguarded
  use arg info, hwnd, id, notifyCode, lb

  LBN_SELCHANGE = 1

  say 'Control ID:          ' id
  say 'Notification code:   ' notifyCode
  say 'Dialog control:      ' lb

  if notifyCode == LBN_SELCHANGE then do
    say 'Current selection is:' lb~selected
  end
  say


/* Output might be:

Control ID:           1003
Notification code:    4
Dialog control:       a ListBox

Control ID:           1003
Notification code:    1
Dialog control:       a ListBox
Current selection is: Bakersfield

Control ID:           1003
Notification code:    1
Dialog control:       a ListBox
Current selection is: San Jose

Control ID:           1003
Notification code:    1
Dialog control:       a ListBox
Current selection is: New York
```

```
Control ID:            1003
Notification code:     1
Dialog control:        a ListBox
Current selection is: Los Angles

*/
```

4.7.10.1. Connect COMMAND Event Handler

The event handler connected through the *connectCommandEvents* is invoked when ooDialog internally matches a message sent to the *underlying* Windows dialog using the arguments specified to the *connectCommandEvents* method.

The *willReply* argument in the *connectCommandEvents* method determines how the event handler needs to respond to the notification.

```
::method onCommandEvent unguarded
  use arg wParam, lParam, id, notifyCode, controlObj

  return 0
```

Arguments:

This event handling method receives five arguments. The *wParam* and *lParam* arguments have to be retained for backwards compatibility, but the last 3 arguments are really what are useful.

wParam
: The numeric value of the*WPARAM* parameter sent with the window message that was connected through *connectCommandEvents*. This numeric value will contain the resource ID of the control sending the message and the numeric code of the event that caused the notification to be sent. However, the ooDialog framework extracts these values and sends them as the *id* and *notifyCode* arguments.

lParam
: The numeric value of the *LPARAM* parameter sent with the window message that was connected through *connectCommandEvents*. The LPARAM value is always the window handle of the control sending the notification.

id
: The numeric resource id of the dialog control sending the notification.

notifyCode
: The numeric notification code of the event that caused the notification to be sent. It will be a notification code defined by the control. Note that notification codes do not have unique values. For example, the list box LBN_SETFOCUS notification code has the value of 4, as does the button's BN_DISABLED code.

controlObj
: The Rexx dialog control object that represents the underlying dialog control that sent the notification. If an error occurred, this will be the **`.nil`** object. It is very unlikely that an error will occur.

Return:

The command event notification is sent as a WM_COMMAND window message. In general the operating system ignores the return from a WM_COMMAND message. This makes 0 a good value to return. However, the MSDN documentation is the authority on what return value the operating system expects from a particular event notification and the ooDialog programmer should consult that documentation for a specific COMMAND event being connected.

Example

The example displays the 3 different types of combo box controls and shows all the event notifications sent from the combo boxes. The ::constant directive is used to define the numeric value of all notifications a combo box might possibly send and the *connectCommandEvents* method is used to connect all notifications to a single event handler. That event handler decodes each notification and displays it:

```
::class 'ComboBoxTypes' subclass RcDialog

::constant CB_SIMPLE        1001
::constant CB_DROPDOWN      1003
::constant CB_DROPDOWNLIST  1005

::constant  CBN_ERRSPACE        -1
::constant  CBN_SELCHANGE       1
::constant  CBN_DBLCLK          2
::constant  CBN_SETFOCUS        3
::constant  CBN_KILLFOCUS       4
::constant  CBN_EDITCHANGE      5
::constant  CBN_EDITUPDATE      6
::constant  CBN_DROPDOWN        7
::constant  CBN_CLOSEUP         8
::constant  CBN_SELENDOK        9
::constant  CBN_SELENDCANCEL    10

::method init

  forward class (super) continue

  self~connectCommandEvents(IDC_CB_SIMPLE, onEvent, .true)
  self~connectCommandEvents(IDC_CB_DROPDOWN, onEvent, .true)
  self~connectCommandEvents(IDC_CB_DROPDOWNLIST, onEvent, .true)

::method onEvent unguarded
    use arg , , id, code, cb

    say 'id:  ' id
    say 'code:' code
    say 'cb:  ' cb

    select
      when id == self~CB_SIMPLE then cbName = 'The simple combo box'
      when id == self~CB_DROPDOWN then cbName = 'The drop down combo box'
      when id == self~CB_DROPDOWNLIST then cbName = 'The drop down list combo box'
      otherwise cbName = 'An error has occurred'
    end

    select
      when code == self~CBN_ERRSPACE     then msg = 'sent a error space notification
 message.'
      when code == self~CBN_SELCHANGE    then msg = 'sent a selection change notification
 message.'
      when code == self~CBN_DBLCLK       then msg = 'sent a double click notification
 message.'
      when code == self~CBN_SETFOCUS     then msg = 'sent a set focus notification
 message.'
      when code == self~CBN_KILLFOCUS    then msg = 'sent a kill focus notification
 message.'
      when code == self~CBN_EDITCHANGE   then msg = 'sent a edit change notification
 message.'
      when code == self~CBN_EDITUPDATE   then msg = 'sent a edit update notification
 message.'
```

```
      when code == self~CBN_DROPDOWN     then msg = 'sent a drop down notification
 message.'
      when code == self~CBN_CLOSEUP      then msg = 'sent a close up notification
 message.'
      when code == self~CBN_SELENDOK     then msg = 'sent a selection ended ok
 notification message.'
      when code == self~CBN_SELENDCANCEL then msg = 'sent a selection ended cancel
 notification message.'
      otherwise msg = 'processing the event notification.'
    end
    -- End select

    say cbName msg
    say

    return 0

/*  Output might be:

id:   1001
code: 3
cb:   a ComboBox
The simple combo box sent a set focus notification message.

id:   1001
code: 4
cb:   a ComboBox
The simple combo box sent a kill focus notification message.

id:   1003
code: 3
cb:   a ComboBox
The drop down combo box sent a set focus notification message.

id:   1003
code: 7
cb:   a ComboBox
The drop down combo box sent a drop down notification message.

id:   1003
code: 10
cb:   a ComboBox
The drop down combo box sent a selection ended cancel notification message.

id:   1003
code: 8
cb:   a ComboBox
The drop down combo box sent a close up notification message.

id:   1003
code: 10
cb:   a ComboBox
The drop down combo box sent a selection ended cancel notification message.
*/
```

4.7.11. connectDateTimePickerEvent

```
>>--connectDateTimePickerEvent(--id-,-evt--+----------+-+------------+-)-------><
                                           +-,--mName-+ +-,-wilReply-+
```

The *connectDateTimePickerEvent* method connects an *event* notification message from a *DateTimePicker* control to a method in the Rexx dialog.

Arguments:

The arguments are:

id [required]

The resource ID of the date time picker control. May be numeric or *symbolic*.

evt [required]

Exactly one of the following keywords. The keyword specifies the event to be connected. For each event, the documentation for the event handler, *CLOSEUP*), *DATETIMECHANGE*, etc., will contain additional information about the event. Case is not significant:

CLOSEUP	FORMAT	KILLFOCUS
DATETIMECHANGE	FORMATQUERY	SETFOCUS
DROPDOWN	KEYDOWN	USERSTRING

CLOSEUP

Sent by a date and time picker (DTP) control when the user closes the drop-down month calendar. The month calendar is closed when the user chooses a date from the month calendar or clicks the drop-down arrow while the calendar is open. The return value from the event handler is ignored by the operating system for this event.

DATETIMECHANGE

Sent by a date and time picker (DTP) control whenever a change occurs. The return value from the event handler is ignored by the operating system for this event.

DROPDOWN

Sent by a date and time picker (DTP) control when the user activates the drop-down month calendar. The return value from the event handler is ignored by the operating system for this event.

FORMAT

Sent by a date and time picker (DTP) control to request text to be displayed in a *callback* field. The *willReply* argument is ignored for this event, the event handler must *return* a reply.

FORMATQUERY

Sent by a date and time picker (DTP) control to retrieve the maximum allowable size of the string that will be displayed in a *callback* field. The *willReply* argument is ignored for this event, the event handler must *return* a reply.

KEYDOWN

Sent by a date and time picker (DTP) control when the user types in a *callback* field. The *willReply* argument is ignored for this event, the event handler must *return* a reply.

KILLFOCUS

Notifies a date and time picker control's parent window, (which is the dialog window,) that the control has lost the input focus. The return value from the event handler is ignored by the operating system for this event.

SETFOCUS

Notifies a date and time picker control's parent window, (which is the dialog window,) that the control has received the input focus. The return value from the event handler is ignored by the operating system for this event.

USERSTRING
Sent by a date and time picker (DTP) control when a user finishes editing a string in the control. This notification message is only sent by DTP controls that have the CANPARSE style. The *willReply* argument is ignored for this event, the event handler must *return* a reply.

mName [optional]
The name of the method that is to be invoked whenever the specified notification is received from the date time picker control. The programmer defines this method. If this argument is omitted, a method name is automatically generated that consists of the event keyword preceded by **`on`**. For instance, **`onCloseUp`**. If the method name is supplied, it can not be the empty string.

willReply [optional]
The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.true`**.

However, this argument is ignored for the USERSTRING, KEYDOWN, FORMAT, and FORMATQUERY events. If the programmer connects any of these events, the interpreter waits for the returned value from the connected method. That value is then returned to the operating system. This can not be changed.

Return value:
This method returns **`.true`** if the event notification was connected correctly, otherwise **`.false`** .

Remarks:
See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

Details:
Syntax errors are raised when incorrect usage is detected, including the use of an invalid symbolic ID or an unrecognized event keyword.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if the connected event happens.

The underlying dialog receives the DTN_* messages as the notifications for the date time picker events.

Example:
The following example creates a DTP control in the dialog template of a **`UserDialog`** and then connects the drop down and close up events:

```
::method defineDialog

  self~createDateTimePicker(IDC_DTP_REPORT_DATE,  10, 7, 280, 15, "BORDER SHORT
 SHOWNONE")

  self~createPushButton(IDOK, 85, 74, 50, 14, "DEFAULT", "Ok")
  self~createPushButton(IDCANCEL, 140, 74, 50, 14, , "Cancel")

  self~connectDateTimePickerEvent(IDC_DTP_REPORT_DATE, "CLOSEUP", onCloseUp)
  self~connectDateTimePickerEvent(IDC_DTP_REPORT_DATE, "DROPDOWN", onDropDown)
```

4.7.11.1. CloseUp Event Handler

The event handler for the close up event is invoked when the user closes the drop-down month calendar. The month calendar is closed when the user chooses a date from the month calendar or clicks the drop-down arrow while the calendar is open.

The programmer can specify for the interpreter to wait, or not wait, for the return from the event handler by using the *willReply* argument in the *connectDateTimePickerEvent* method. The actual value returned from the event handler is ignored.

This event notification signals that the DTP control has destroyed the child month calendar control. The DTP control creates a new month calendar each time the month calendar needs to be shown and destroys it each time the drop-down is closed. The close up notification is sent when the month calendar is destroyed.

The programmer may have instantiated a Rexx month calendar object during the *DROPDOWN* event notification. Once the underlying Windows month calendar is destroyed, the Rexx **`MonthCalendar`** is no longer valid.

```
::method onCloseUp unguarded
  use arg idFrom, hwndFrom, code, dtpObj

  return 0
```

Arguments:

The event handling method receives r arguments:

idFrom

The *idFrom* argument is the resource ID of the date time picker that generated the notification.

hwndFrom

The window handle of the date and time picker.

code

The date time picker event code the caused the event notification to be sent.

dtpObj

The Rexx date time picker object that represents the date time picker control that sent the notification.

Return:

The return value from the event handler is ignored by the operating system. 0 makes a good return value.

Example

The following is an example event handler for the close up event. When the close up happens, the user may or may not have changed the selected date. The example checks if a new data has been selected and does something if it has been changed:

```
::method onCloseUp unguarded
  expose lastChange
  use arg id, hwnd, code, dtp

  newDate = dtp~getDateTime
  if newDate \= lastChange then do
    lastChange = newDate
    self~updateDateInfo(newDate)
  end

  return 0
```

4.7.11.2. DateTimeChange Event Handler

The event handler for the date time change event is invoked whenever a change in the underlying DTP control takes place

The programmer can specify for the interpreter to wait, or not wait, for the return from the event handler by using the *willReply* argument in the *connectDateTimePickerEvent* method. The actual value returned from the date time change event handler is ignored.

```
::method onDateTimeChange unguarded
  use arg dateTime, valid, idFrom, hwndFrom, code, dtpObj

  return 0
```

Arguments:

The event handling method receives 6 arguments:

dateTime

The *dateTime* argument is a **`DateTime`** object. When the *valid* argument is true, *dateTime* specifies the new displayed date and time in the DTP control. When *valid* is false, *dateTime* will be the exact date and time that the notification was received.

valid

True or false indicating whether the date time change is valid or not. *valid* can only be false when the DTP control has the SHOWNONE style and the user has unchecked the check box to indicate that no date and time is currently selected.

idFrom

The *idFrom* argument is the resource ID of the date time picker that generated the notification.

hwndFrom

The window handle of the date and time picker.

code

The date time picker event code the caused the event notification to be sent.

dtpObj

The Rexx date time picker object that represents the date time picker control that sent the notification.

Return:

The return from the event handler is ignored. Returning 0 is sensible.

Example

The following example updates a static control with the currently selected date and time whenever the user changes the date and time:

```
::method initDialog
  expose dtp staticMsg

  self~connectDateTimePickerEvent(IDC_DTP, "DATETIMECHANGE", onChange)

  dtp = self~newDateTimePicker(IDC_DTP);
  dtp~setFormat("hh':'mm':'ss dddd MMM dd', 'yyyy")

  staticMsg = self~newStatic(IDC_ST1)
  ...
```

```
::method onChange unguarded
  expose staticMsg
  use arg dateTime, valid, idFrom, hwndFrom

  if valid then do
    sf = .SimpleFormatter~new(dateTime)
    staticMsg~setText(sf~time 'on' sf~date)
  end
  else do
    staticMsg~setText("No valid date selected.")
  end

  return 0

::class 'SimpleFormatter' public

::method init
  expose dateTime
  use strict arg dateTime

::method date
  expose dateTime

  dayNum = dateTime~usaDate~substr(4, 2)~strip('L', '0')
  year = dateTime~standardDate~left(4)

  return dateTime~dayName || ',' dateTime~monthName dayNum || ',' year

::method time
  expose dateTime

  return dateTime~civilTime
```

4.7.11.3. DropDown Event Handler

The event handler for the drop down event is invoked when the user activates the drop down month calendar. The DTP control sends the notification for the event after it has created the child month calendar control.

The programmer can specify for the interpreter to wait, or not wait, for the return from the event handler by using the *willReply* argument in the *connectDateTimePickerEvent* method. The actual value returned from the event handler for the drop down event is ignored.

```
::method onDropDown unguarded
  use arg idFrom, hwndFrom, code, dtpObj

  return 0
```

Arguments:

The drop down event handler receives 4 arguments:

idFrom

The *idFrom* argument is the resource ID of the date time picker that generated the notification.

hwndFrom

The window handle of the date and time picker.

code

The date time picker event code the caused the event notification to be sent.

dtpObj
: The Rexx date time picker object that represents the date time picker control that sent the notification.

Return:

The return from the event handler is ignored. Returning 0 is sensible.

Remarks:

One reason for handling the drop down event is to adjust the style of the month calendar control when it is displayed. Note that the DTP control creates the child month calendar control when it is needed and destroys the month calendar when the drop down is closed up. Therefore you can not save a reference to the month calendar object and use it later. When the *underlying* month calendar control is destroyed the Rexx month calendar object is no longer valid.

In Windows Vista and later versions of Windows, the DTP control has the *setMonthCalStyle* method which can be used to set the month calendar's style once and the DTP control will then use that style each time it creates the month calendar. Therefore, the technique of setting the month calendar style during the drop down event handler is really only needed for Windows XP and earlier.

Example

The following example demonstrates how to change the month calendar style in the drop down event handler. If the operating system the program is running on is Vista or later, the whole process is bypassed:

```
::method initDialog
  dtp = self~newDateTimePicker(IDC_DTP_APPOINTMENT_TIME);

  if .OS~isAtLeastVista then do
    dtp~setMonthCalStyle("NOCIRCLE NOTRAILING")
  end
  else
    self~connectDateTimePickerEvent(IDC_DTP_APPOINTMENT_TIME, "DROPDOWN", onDropDown)
  do
  ...

::method onDropDown unguarded
  use arg idFrom, hwndFrom, code, dtp

  -- We know this is not Vista or later, othewise we would not
  -- have gotten the event notification.

  mc = dtp~getMonthCal
  mc~replaceStyle('DAYSTATE MULTI WEEKNUMBERS', "NOCIRCLE NOTODAY")

  return 0
```

4.7.11.4. Format Event Handler

The event handler for the format event is invoked when the DTP control requests the text to be displayed in a *callback* field.

The programmer must return the text to display for the callback field and the interpreter waits for this return. This behaviour can not be changed.

```
::method onFormat unguarded
  use arg field, dateTime, idFrom, hwndFrom, code, dtpObj
```

```
return text
```

Arguments:

The event handling method receives 6 arguments:

field

The text of the callback field identifier.

dateTime

A **`DateTime`** object that is the currently displayed date and time in the DTP control.

idFrom

The *idFrom* argument is the resource ID of the date time picker that generated the notification.

hwndFrom

The window handle of the date and time picker.

code

The date time picker event code the caused the event notification to be sent.

dtpObj

The Rexx date time picker object that represents the date time picker control that sent the notification.

Return:

The text to display for the specified callback field. The text must be no longer than 63 characters or a syntax condition is raised.

Example

The **`FORMAT`**, *FORMATQUERY*, and *KEYDOWN* event notifications all work together to provide the functionality for call back fields. It is a little difficult to grasp the concepts from small snippets of code, so a complete example program is included in the ooDialog samples: **`samples\oodialog\controls\fiscalReports.rex`**.

This is an example FORMAT event handler. It comes from the complete **`fiscalReports.rex`** program:

```
::method onFormat unguarded
  expose periods types currentType currentPeriod
  use arg field, dt, id, hwnd

  select
    when field == 'XX' then do
      ret = self~getPeriodNumber(dt)
    end

    when field == 'XXX' then do
      ret = periods[currentPeriod]
    end

    otherwise do
      ret = types[currentType]
    end
  end
  -- End select

  return ret
```

4.7.11.5. FormatQuery Event Handler

The event handler for the format query event is invoked when the DTP control requests the maximum size needed to display a string in a *callback* field.

The programmer must fill in a *Size* object with the maximum size needed and return a value from the event handler. The interpreter waits for this return. This behavior can not be changed.

```
::method onFormatQuery unguarded
  use arg field, size, id, hwnd, code, dtpObj

  return 0
```

Arguments:

The event handling method receives 6 arguments:

field
: The text of the callback field identifier.

size
: A **`Size`** object that is set to the maximum size required to display the string in the call back field.

idFrom
: The *idFrom* argument is the resource ID of the date time picker that generated the notification.

hwndFrom
: The window handle of the date and time picker.

code
: The date time picker event code the caused the event notification to be sent.

dtpObj
: The Rexx date time picker object that represents the date time picker control that sent the notification.

Return:

The actual value returned from the event handler is ignored by the operating system. Rather the return signals the operating system that the *size* argument is now valid to access, and the OS can use its values. Returning zero is sensible.

Example

The **`FORMATQUERY`**, *FORMAT*, and *KEYDOWN* event notifications all work together to provide the functionality for call back fields. It is a little difficult to grasp the concepts from small snippets of code, so a complete example program is included in the ooDialog samples: **`samples\oodialog\controls\fiscalReports.rex`**.

The following example FORMATQUERY event handler comes from that example program:

```
::method onFormatQuery unguarded
  expose haveSizes xxSize xxxSize xxxxSize
  use arg field, size, id, hwnd, code, dtp

  if \ haveSizes then do
    xxSize  = self~calcSize('XX')
    xxxSize = self~calcSize('XXX')
    xxxxSize = self~calcSize('XXXX')
    haveSizes = .true
  end

  -- The equateTo() method sets the cx and cy attributes of the receiver .Size
  -- object to the cx and cy attributes of the argument .Size object.
  select
    when field == 'XX' then size~equateTo(xxSize)
    when field == 'XXX' then size~equateTo(xxxSize)
    otherwise size~equateTo(xxxxSize)
  end
  -- End select

  return 0
```

4.7.11.6. KeyDown Event Handler

The event handler for the key down event is invoked when the user types a key in a *callback* field.

The programmer must return a **`DateTime`** object and the interpreter waits for this return. This behavior can not be changed.

```
::method onKeyDown unguarded
  use arg field, dateTime, vKey, idFrom, hwndFrom, code, dtpObj

  return dateTime
```

Arguments:

The event handling method receives 7 arguments:

field
: The text of the callback field identifier.

dateTime
: A **`DateTime`** object that is the currently displayed date and time in the DTP control.

vKey
: The virtual key code of the key the user typed in the call back field. The *VK* class can be used to map the numeric key code to a symbol, making it easier to work with the codes.

idFrom
: The *idFrom* argument is the resource ID of the date time picker that generated the notification.

hwndFrom
: The window handle of the date and time picker.

code
: The date time picker event code the caused the event notification to be sent.

dtpObj
: The Rexx date time picker object that represents the date time picker control that sent the notification.

Return:

The programmer returns a **`DateTime`** object. If the return is a date / time different than the *dateTime* argument, the DTP control's date / time is updated. If it is he same, then no action is taken by the DTP control.

Remarks:

By examining the *dateTime* and *vKey* arguments the programmer can produce custom responses to the user's typed keys. The custom response is achieved by returning a **`DateTime`** object that specifies a different date than the currently displayed date. For instance, if the currently displayed date is 8:00 am July 4th 1998 and the user types the *home* key, the programmer could return a date of 12:00 am January 1st 1998 to set the DTP control's display to the first of January in 1998. Likewise, if the user typed the *end* key the programmer could return December 31 1998 at 11:59 pm to set the display to the end of the current year.

Example

The **`KEYDOWN`**, *FORMAT*, and *FORMATQUERY* event notifications all work together to provide the functionality for call back fields. It is a little difficult to grasp the concepts from small snippets of code, so a complete example program is included in the ooDialog samples: **`samples\oodialog\controls\fiscalReports.rex`**.

The following example shows the KEYDOWN event handler from the **`fiscalReports.rex`** example program:

```
::method onKeyDown unguarded
  use arg field, dt, vKey, idFrom, hwndFrom, code, dtpObj

  select
    when field == 'XX' then do
      newDT = self~updatePeriodNumber(dt, vKey)
    end

    when field == 'XXX' then do
      newDT = self~updatePeriod(dt, vKey)
    end

    otherwise do
      newDT = self~updateReport(dt, vKey)
    end
  end
  -- End select

  return newDT
```

4.7.11.7. KillFocus Event Handler

The event handler for the kill focus event is invoked when the DTP control loses the input focus.

The programmer can specify for the interpreter to wait, or not wait, for the return from the event handler by using the *willReply* argument in the *connectDateTimePickerEvent* method. The actual value returned from the event handler for the kill focus event is ignored.

```
::method onKillFocus unguarded
  use arg idFrom, hwndFrom, code, dtpObj

  return 0
```

Arguments:

The kill focus event handler receives 4 arguments:

idFrom

The *idFrom* argument is the resource ID of the date time picker that generated the notification.

hwndFrom
: The window handle of the date and time picker.

code
: The date time picker event code the caused the event notification to be sent.

dtpObj
: The Rexx date time picker object that represents the date time picker control that sent the notification.

Return:

The return from the event handler is ignored. Returning 0 is sensible.

4.7.11.8. SetFocus Event Handler

The event handler for the set focus event is invoked when the DTP control gains the input focus.

The programmer can specify for the interpreter to wait, or not wait, for the return from the event handler by using the *willReply* argument in the *connectDateTimePickerEvent* method. The actual value returned from the event handler for the set focus event is ignored.

```
::method onSetFocus unguarded
  use arg idFrom, hwndFrom, code, dtpObj

  return 0
```

Arguments:

The set focus event handler receives 4 arguments:

idFrom
: The *idFrom* argument is the resource ID of the date time picker that generated the notification.

hwndFrom
: The window handle of the date and time picker.

code
: The date time picker event code the caused the event notification to be sent.

dtpObj
: The Rexx date time picker object that represents the date time picker control that sent the notification.

Return:

The return from the event handler is ignored. Returning 0 is sensible.

4.7.11.9. UserString Event Handler

The event handler for the USERSTRING event is invoked when the user has finished editing a string in the DTP control. This event notification only occurs when the DTP control has the CANPARSE style.

The programmer must return a value from the event handler and the interpreter waits for this return. This behavior can not be changed.

```
::method onUserString unguarded
  use arg dateTime, userStr, idFrom, hwndFrom, code, dtpObj
```

```
return newDateTime
```

Arguments:

The event handling method receives 6 arguments:

dateTime
: A **`DateTime`** object that reflects the date and time the DTP control is currently displaying.

userStr
: The string the user has just finished typing into the DTP control.

idFrom
: The *idFrom* argument is the resource ID of the date time picker that generated the notification.

hwndFrom
: The window handle of the date and time picker.

code
: The date time picker event code the caused the event notification to be sent.

dtpObj
: The Rexx date time picker object that represents the date time picker control that sent the notification.

Return:

The event handler must return a **`DateTime`** object or the **`.nil`** object.

After parsing the user string, the programmer returns a new **`DateTime`** object that reflects the new date and time the DTP control should display. If the returned **`DateTime`** object reflects the exact same date and time as the *dateTime* argument, then the DTP control takes no action. Otherwise, the DTP control updates its display to the new date and time.

If, and only if, the DTP control has the SHOWNONE style, the programmer can return the **`.nil`** object to change the date and time to *no date and time* selected.

Remarks:

This event notification allows the programmer to provide a custom response to what the user types into the DTP control's display field. The programmer parses the string entered by the user and then updates the DTP control in a way that corresponds to the entered string.

Example

The distribution of the ooDialog framework contains an example program: **`samples\oodialog\controls\userStringDTP.rex`** that focuses on the USERSTRING notification. This example is a portion of the event handler for the USERSTRING event from that program.

The DTP control in the program is initially set to the current system date and time. Naturally, as the program runs, the system date and time continue to advance. It would be difficult for the user to reset the DTP control to the current system date and time, so the program allows the user to type a *r* or a *c* in the DTP to reset its display to the current system date and time:

```
::method onUserString unguarded
  expose resetting stInvalid
  use arg dt, userStr, id, hwnd, code, dtpObj

  stInvalid~setText('')
```

```
-- Check for the shortcut to set the DTP to the current date and time.
upStr = userStr~upper
if upStr == 'C' | upStr == 'R' | upStr == 'CANCEL' | upStr == 'RESET' then do
  resetting = .true
  return .DateTime~new
end

...

return newDT
```

4.7.12. connectDraw

```
>>--connectDraw--(--+-----+--+---------------+--+--------------+--)--------------><
                    +--id-+  +-,-methodName--+  +-,-willReply--+
```

The *connectDraw* method connects the draw control event notification with a method in the Rexx dialog. This notification is sent to the underlying dialog by an owner-drawn button, combo box, list box, static, or menu, when a visual aspect of the control or menu has changed. In addition, if a tab control or a list-view have the owner draw fixed style, they also receive the notification.

Arguments:

The arguments are:

id [optional]

The resource ID of the dialog control whose notification is being connected. This can be symbolic or numeric. If the ID is omitted, all drawing event notifications, of all owner-drawn controls in the dialog, will invoke the method.

methodName [optional]

The name of the method that is to be invoked each time the draw control event occurs. The method name must not be the empty string. If this argument is omitted, then the ooDialog framework will connect the notification to the **`onDraw`** method.

willReply [optional]

The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.false`**.

Return value:

-1

The specified symbolic ID could not be resolved.

0

No error.

1

The notification was not connected correctly.

Remarks:

The notification for the draw control is only sent to the above mentioned controls when they have the OWNERDRAW or OWNERDRAWFIXED styles. The notification itself is to inform the application that it needs to draw the control at this time. Note that the ooDialog framework is not well suited to drawing operations and it is unlikely that this notification is of much use except when the drawing to be done is not very complex.

Prior to ooDialog 4.2.4, the arguments sent to the event handler for this connection were not helpful in attempting to do owner draw. The arguments now sent to the event handler will make the task easier and ambitious programmers may want to try owner draw in medium complex applications. **Note** however that the *willReply* argument must be set to **`.true`** or *SYNC* or the drawing efforts will produce highly unpredictable results.

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

Details:

Syntax errors are raised when incorrect usage is detected.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if any draw events happen.

In Windows itself, the dialog receives this event notification as a WM_DRAWITEM message.

4.7.12.1. Draw Event Handler

The event handler for the Draw event is invoked when a button, combo box, list box, or menu, with the owner-draw style, needs to be redrawn.

The *willReply* argument in the *connectDraw* method determines how the event handler needs to respond to the notification.

```
::method onDraw unguarded
  use arg id, lp, drawObj, itemID, flags, hDC, rcItem, itemData

  return .true
```

Arguments:

The event handling method receives 8 arguments:

id

The resource ID of the control that needs to be drawn. This value is not valid for menu items

lp

The numeric value of a pointer to a memory location. This is of no use to the ooDialog programmer. There is no way to access the data the pointer points to from the ooDialog program. This argument is retained for backwards compatibility. The following arguments contain the values of the data the pointer points to. They are the important arguments.

drawObj

The Rexx dialog control object that represents the control that needs to be drawn. If it is a menu item that needs to be drawn, this may be the **`.nil`** object. **Please Note:** In future versions of ooDialog this may be the Rexx menu object that contains the menu item needing to be drawn.

itemID

The menu item ID for a menu item or the one-based index of the item in a list box or combo box. For an empty list box or combo box, this member can be 0. This allows the application to draw only the focus rectangle at the coordinates specified by the rcItem member even though there are no items in the control. This indicates to the user whether the list box or combo box has the focus. Which keywords are present in the *flags* determines whether the rectangle is to be drawn as though the list box or combo box has the focus.

flags

A list of the following keywords separated by spaces, case is not significant. The keywords are separated into 3 groups, type, action, and state. The keywords in the same group all start with the same prefix, ODT_ for type, ODA_ for action, and ODS_ for state. The list will contain at least one keyword from the type group. The keywords provide information that allows the application to determine what drawing operation needs to be done:

ODT_BUTTON	ODT_UNKNOWN	ODS_FOCUS
ODT_COMBOBOX	ODA_DRAWENTIRE	ODS_GRAYED
ODT_HEADER	ODA_FOCUS	ODS_HOTLIGHT
ODT_LISTBOX	ODA_SELECT	ODS_INACTIVE
ODT_LISTVIEW	ODS_CHECKED	ODS_NOACCEL
ODT_MENU	ODS_COMBOBOXEDIT	ODS_NOFOCUSRECT
ODT_STATIC	ODS_DEFAULT	ODS_SELECTED
ODT_TAB	ODS_DISABLED	

ODT_BUTTON

An owner-drawn button is sending the notification.

ODT_COMBOBOX

A owner-drawn combo box is sending the notification.

ODT_HEADER

A header control is sending the notification. It is unlikely this keyword will ever be seen. ooDialog does not yet have support for header controls and MSDN does not document this.

ODT_LISTBOX

An owner-drawn list box is sending the notification.

ODT_LISTVIEW

A list-view is sending the notification.

ODT_MENU

An owner-drawn menu item is sending the notification.

ODT_STATIC

An owner-drawn static control is sending the notification.

ODT_TAB

A tab control is sending the notification.

ODT_UNKNOWN

The notification was sent by something unexpected to the ooDialog framework. It is doubtful that this keyword will ever be seen.

ODA_DRAWENTIRE

The entire control needs to be drawn.

ODA_FOCUS

The control has lost or gained the keyboard focus. The state keywords should be checked to determine whether the control has the focus.

ODA_SELECT

The selection status has changed. The state keywords should be checked to determine the new selection state.

ODS_CHECKED
: The menu item is to be checked. This keyword is used only in a menu.

ODS_COMBOBOXEDIT
: The drawing takes place in the selection field (edit control) of an owner-drawn combo box.

ODS_DEFAULT
: The item is the default item.

ODS_DISABLED
: The item is to be drawn as disabled.

ODS_FOCUS
: The item has the keyboard focus.

ODS_GRAYED
: The item is to be grayed. This keyword is used only in a menu.

ODS_HOTLIGHT
: The item is being hot-tracked, that is, the item will be highlighted when the mouse is on the item.

ODS_INACTIVE
: The item is inactive and the window associated with the menu is inactive.

ODS_NOACCEL
: The control is drawn without the keyboard accelerator cues.

ODS_NOFOCUSRECT
: The control is drawn without focus indicator cues.

ODS_SELECTED
: The menu item's status is selected.

hDC
: The *device context* for the drawing area. This device context must be used to do the drawing. It is passed by the operating system and *must not* freed by the application. In addition, upon return from the event handler, the device context must be set back to the same state it was on entry to the event handler.

rcItem
: A *Rect* object that defines the boundaries of the control to be drawn. This rectangle is in the device context specified by the *hDC* argument. The operating system automatically clips anything that the owner window draws in the device context for combo boxes, list boxes, and buttons, but does not clip menu items. When drawing menu items, the owner window must not draw outside the boundaries of the rectangle defined by this argument.

itemData
: The item data associated with the menu item or control, or the **`.nil`** object when there is no item data. Menu items and many dialog controls allow the application to associate a user object with each item. However, ooDialog does not completely support this at this time. For instance, the *getItemData* and *setItemData* methods support associating user objects with the list-view items. But the support has not yet been added to combo boxes and list boxes. Because of this, it is likely that the *itemData* argument will usually be the **`.nil`** object.

Return:

The MSDN documentation says that the event handler should return true when the application has handled the DRAW event notification. Which implies that false should be returned otherwise. Therefore, the ooDialog programmer should probably return true from the event handler. However, some experimentation with returning false has not shown any discernable difference than returning true.

Remarks:

Although this documentation says the device context passed in the *hDC* argument must be used to do the drawing in the event handler, ooDialog has examples that did not do this. Examples written prior to ooDialog 4.2.0 did not have the device context passed into the event handler. These examples used the *getControlDC* method to get a device context for the owner-drawn control. This technique seems to work okay. Nevertheless, if the ooDialog programmer is trying to use owner-drawn controls, it is highly likely that performing the user drawing the way it is intended to be done is more likely to have success.

When the device context passed as the *hDC* argument is used to perform the drawing, as intended by the operating system, it is critical that the *willReply* argument be set to true or *SYNC*. The result of any drawing operations done with the device context when the interpreter does not wait for the event handler to return will be highly unpredictable.

Example

The following example creates a large owner-drawn button that takes up most of the dialog. The client area of the button is then used as a *canvass* to draw on. The event handler is invoked every time the button needs to be drawn. The event handler draws a circle on the button area and writes *Circle* on the button. This is a complete example that allows the user to experiment a little. An interesting experiment is to uncomment the say statement in the *drawIt* method to see when the event handler is executing:

```
  dlg = .OwnerDrawDlg~new
  if dlg~initCode <> 0 then return 99
  dlg~execute("SHOWTOP")

return 0

::requires "ooDialog.cls"

::class 'OwnerDrawDlg' subclass UserDialog

::method init

  forward class (super) continue
  self~create(6, 15, 187, 135, "Owner-drawn Button Dialog", "CENTER")

  self~connectDraw(10, "drawIt", .true)

::method defineDialog
  self~createPushButton(10, 6, 6, 175, 123, "OWNER", "")

::method drawIt unguarded
  use arg id, lp, drawObj, itemID, flags, dc, r, itemData

  -- Uncomment this line to seen when the event handler is executed:
  --say 'Draw It'

  -- Create a pen to draw a circle with.
  pen    = drawObj~createPen(5, "SOLID", 1)
  oldPen = drawObj~objectToDc(dc, pen)

  -- Create a font to use to draw text with.
```

```
properties = .directory~new
properties~weight = 700
properties~italic = .true

font    = drawObj~createFontEx("Arial", 24, properties)
oldFont = drawObj~fontToDC(dc, font)

drawObj~transparentText(dc)

-- Get the midpoint of the button rectangle and set the text align to center
pos = .Point~new(r~right % 2, r~bottom  % 2)
oldAlign = drawObj~setTextAlign(dc, 'CENTER BASELINE NOUPDATECP')

-- Get a rectangle indented 5 from the button's area.
dr = .Rect~new(r~left + 5, r~top + 5, r~right - 10, r~bottom - 10)

-- Draw a circle, within the rectangle
drawObj~drawArc(dc, dr~left, dr~top, dr~right, dr~bottom)

-- Write some text at the position we calculated above.
drawObj~writeDirect(dc, pos~x, pos~y, 'Circle')

-- Now restore the DC so it is the same as passed into us.
drawObj~setTextAlign(dc, oldAlign)
drawObj~fontToDC(dc, oldFont)
drawObj~objectToDc(dc, oldPen)

drawObj~opaqueText(dc)

drawObj~deleteFont(font)
drawObj~deleteObject(pen)

return .true
```

4.7.13. connectEditEvent

```
>>--connectEditEvent(--id--,--event--+----------+--+--------------+--)---------><
                                     +-,-mName--+  +-,-willReply--+
```

Connects a method in the Rexx dialog to the Windows *event* notification from a *Edit* control.

Arguments:

The arguments are:

id [required]

The ID of the edit control whose notification event is to be connected to a method.

event [required]

A single keyword, case is not significant, specifying the event to be connected with a method:

CHANGE

The text has been altered. This notification is sent after the screen has been updated.

ERRSPACE

An out-of-memory problem has occurred.

GOTFOCUS

The edit control got the input focus.

HSCROLL
The horizontal scroll bar has been used.

LOSTFOCUS
The edit control lost the input focus.

MAXTEXT
The text inserted exceeds the specified number of characters for the edit control. This notification is also sent when:

- An edit control does not have the ES_AUTOHSCROLL or AUTOSCROLLH style and the number of characters to be inserted would exceed the width of the edit control.
- The ES_AUTOVSCROLL or AUTOSCROLLV style is not set and the total number of lines resulting from a text insertion would exceed the height of the edit control.

UPDATE
The text has been altered. This notification is sent before the screen is updated.

VSCROLL
The vertical scroll bar has been used.

mName [optional]
The name of the method to invoke whenever the specified notification is received from the edit control. Provide a method with a matching name. If you omit this argument, a method name is generated automatically. The name consists of the event keyword preceded by **on**. For instance: **onLostFocus**. The method name can not be the empty string.

willReply [optional]
The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **.false**.

Return value:

The return codes are:

0
No error detected.

-1
The resource ID could not be resolved or the event argument is incorrect.

1
The message was not connected correctly.

Remarks:

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

Example:

The following example verifies the input of the edit control with the symbolic ID of IDC_EDIT_AMOUNT and resets it to 0 if a nonnumeric value was entered:

```
::class MyDlgClass subclass UserDialog

::method init
  self~init:super(...)
  self~connectEditEvent("IDC_EDIT_AMOUNT", "CHANGE")
```

```
::method onChange unguarded
  ec = self~newEdit("IDC_EDIT_AMOUNT")
  if ec~getText~space(0) \== "" & ec~getText~dataType("N") == 0 then do
    ec~setModified(.false)
    ec~select
    ec~replaceSelText("0")
  end
  return 0
```

4.7.13.1. General Edit Event Handler

The event handler for all of the edit control events receives the same arguments and is coded in the same fashion. When the handler is invoked is fairly clear from the name of the event. The GOTFOCUS event handler is invoked when the edit control gets the focus, etc..

The *willReply* argument in the *connectEditEvent* method determines how the event handler needs to respond to the notification.

```
::method onEditEvent unguarded
  use arg info, hwnd, id, notifyCode, controlObj

  return 0
```

Arguments:

The event handling method receives 5 arguments. The first and second arguments need to be retained for backwards compatibility, but only the last 3 arguments are really needed:

info

A numeric value that contains info about the event. The low order word contains the resource ID of the edit control sending the event. The high order word contains the edit control event code. The *loWord* and *hiWord* methods of the *DlgUtil* class can be used to extract these values. However, the ooDialog framework now extracts those values for you and sends them as the third and fourth arguments.

hwnd

The window handle of the edit control that sent the notification.

id

The numeric resource id of the edit control sending the notification.

notifyCode

The numeric notification code of the event that caused the notification to be sent. Each dialog control has its own specific notification codes.

controlObj

The Rexx edit control object that represents the underlying dialog control that sent the notification. It is possible this will be the **`.nil`** object if some error happened, but this is very unlikely.

Return:

The operating system ignores the return from this notification. 0 makes a good return value.

Example

The following example connects several edit controls to the same event handle and then uses the *id* argument to determine which edit control is sending the notification and take some action specific to that edit control. The code is somewhat simplistic as it ignores what the notification code is. In a real word program the actions taken would depend on what the event was:

```
::method onEditEvent unguarded
  use arg info, hwnd, id, notifyCode, editObj

  if id == .constDir[IDC_EDIT_AMOUNT] then do
    if editObj~getText~space(0) \== "" & editObj~getText~dataType("N") == 0 then do
      editObj~setModified(.false)
      editObj~select
      editObj~replaceSelText("0")
    end
  end
  else if id == .constDir[IDC_EDIT_ZIP] then do
    zip = editObj~getText~strip
      if \ self~validZipCode(zip) then doe
        -- Put up message box telling
        -- user what the problem is ...
        editObj~setText("")
      end
  end

  ...

  return 0
```

4.7.14. connectHelp

```
>>--connectHelp(--+-------------+--+-------------+--)---------><
                  +--methodName--+  +-,-willReply--+
```

The *connectHelp* method connects the Windows Help event with a method in the dialog. The Windows Help event occurs when the user presses the F1 key. (Only the Help events generated when the dialog is the active window are connected.)

Arguments:

The arguments are:

methodName [optional]

The name of the method that to be invoked when the help event occurs. The name can not be the empty string. When this argument is omitted the name defaults to *onHelp*.

willReply [optional]

The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.false`**.

Return value:

0

No error.

1

An (internal) error prevented the message from being connected to a method.

Remarks:

Note that the Windows help event notification connected by this method is not the same as the help **command** event notification *automatically* connected when a dialog object is instantiated.

Details:

Syntax errors are raised when incorrect usage is detected.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if any help events happen.

In Windows itself, the dialog receives this notification as a WM_HELP message.

Example:

```
::method init
    self~init:super
    ...
    self~connectResize(onResize)
    self~connectHelp(onHelp)
    ...

::method onHelp unguarded
    use arg id, type, mouseX, mouseY, cntxID, helpObj
    if type == "MENU" then w = 'Menu id' id; else w = 'Dialog id' id
    say "Help request:"
    say " " w
    say "  Mouse position x:" mouseX "y:" mouseY 'help object:' helpObj

    return 0

/* As the user presses the F1 key at various times when the dialog has the focus
 * the output might be as follows:
 */

Help request:
  Dialog id 12
  Mouse position x: 420 y: 106 help object: a Button
Help request:
  Menu id 60
  Mouse position x: 204 y: 93 help object: the NIL object
Help request:
  Menu id 65
  Mouse position x: 203 y: 166 help object: the NIL object
Help request:
  Dialog id 14
  Mouse position x: 218 y: 410 help object: a RadioButton
Help request:
  Dialog id 80
  Mouse position x: 387 y: 462 help object: a Tab
```

4.7.14.1. Help Event Handler

The event handler for the Help event is invoked when the user presses the F1 key when the active window is the dialog.

The *willReply* argument in the *connectHelp* method determines how the event handler needs to respond to the notification.

```
::method onHelp unguarded
  use arg id, type, mouseX, mouseY, cntxID, helpObj

  return .true
```

Arguments:

The event handling method receives 6 arguments:

id
: The resource ID of the dialog, dialog control, or menu item that had the focus when the F1 key was pressed.

type
: Specifies if the ID in argument 1 was from a window (a dialog or dialog control) or from a menu item. This argument will either be **`WINDOW`** or **`MENU`**.

mouseX
: The x coordinate of the mouse at the time the F1 key was pressed. This value is an absolute screen coordinate (pixel.) Note that the mouse will not necessarily be over the dialog.

mouseY
: The y coordinate of the mouse at the time the F1 key was pressed. The same caveats as for the *mouseX* argument apply.

cntxID
: The help context identifier of the dialog, control, or menu. ooDialog does not yet have good support for setting help context IDs for dialogs or dialog controls. However, ooDialog does support setting the help ID for menus. If the *type* argument is *WINDOW*, this argument will probably be 0.

helpObj
: The Rexx dialog object or the Rexx dialog control object that had the focus when the F1 key was pressed. Otherwise, the **`.nil`** object if it was a menu item that had the focus. **Note:** It is anticipated that future versions of ooDialog will return the Rexx menu object that contains the menu item instead of the **`.nil`** object.

Return:

The MSDN documentation says to return true from the event handler.

Example

The following example shows a simple event handler that prints out the values of the arguments sent to it. This can be useful is understanding how the event handler for the *connectHelp* method works:

```
::method init
  expose menubar

  forward class (super) continue

  if \ self~createMenuBar then do
    self~initCode = 1
    return
  end

  self~connectHelp('onHelp', 'SYNC')

::method onHelp unguarded
  use arg id, type, x, y, cntxID, helpObj

  say 'onHelp'
  say 'id:' id 'type:' type 'x:' x 'y:' y 'cntxID:' cntxID 'helpObj:' heldObj

  return .true

/* Output might be for example:
onHelp
id: 1 type: WINDOW x: 796 y: 463 cntxID: 0 helpObj: a Button
```

```
onHelp
id: 2 type: WINDOW x: 796 y: 463 cntxID: 0 helpObj: a Button
onHelp
id: 1002 type: WINDOW x: 796 y: 463 cntxID: 0 helpObj: a Tab
onHelp
id: 65535 type: WINDOW x: 796 y: 463 cntxID: 0 helpObj: a GroupBox
onHelp
id: 1003 type: WINDOW x: 796 y: 463 cntxID: 0 helpObj: a RadioButton
onHelp
id: 833 type: MENU x: 976 y: 319 cntxID: 46 helpObj: The NIL object
onHelp
id: 821 type: MENU x: 976 y: 319 cntxID: 45 helpObj: The NIL object
onHelp
id: 821 type: MENU x: 323 y: 161 cntxID: 45 helpObj: The NIL object
onHelp
id: 812 type: MENU x: 323 y: 161 cntxID: 44 helpObj: The NIL object
onHelp
id: 1003 type: WINDOW x: 796 y: 375 cntxID: 0 helpObj: a RadioButton
onHelp
id: 1003 type: WINDOW x: 473 y: 347 cntxID: 0 helpObj: a RadioButton

*/
```

4.7.15. Connecting Key Press Events

This section contains methods related to connecting key press events.

4.7.15.1. connectFKeyPress

```
>>--connectFKeyPress(--methodName--)-------------><
```

The *connectFKeyPress* method connects a function key press event notification to a method in the Rexx dialog object.

Arguments:

The single arguments is;

methodName [required]

The name of the method that is to be invoked when the key press event happens. The argument can not be the empty string.

Return value:

The possible return values are:

0

Success.

-2

The underlying mechanism in the Windows API that is used to capture key events failed.

-6

The maximum number of connections has been reached.

-7

The *methodName* method is already connected to a key down event for this dialog.

Remarks

This method works for function keys F2 through F24. In Windows the F1 key is the help key and the *connectHelp* method should be used for F1. This method is a convenience method and is exactly equivalent to:

```
::method initDialog
  ...
  keys = .VK~F2 "-" .VK~F24
  self~connectKeyPress(methodName, keys)
```

Both the *connectFKeyPress* and the *connectKeyPress* methods use the same event *handler*.

Unlike most other methods that connect event notifications, the underlying Windows dialog must exist before this method can be used. That means it can be used in *initDialog* or any time thereafter. There is a maximum limit of 63 methods, per dialog, that can be connected to key press events. Connections can be removed using the *disconnectKeyPress* method if there is no longer a need for a notification of a key press.

The dialog control object also has a *connectFKeyPress*() method. The method of the dialog object (this method) will capture any F key press event when the dialog is the active window. The method of the dialog control object will only capture a F key press when the control has the keyboard focus.

Due to the nature of key press events, the low-level implementation of capturing the key strokes is different from most of the other methods of the **`EventNotification`** class. There is no single message sent to the underlying dialog for a key stroke event.

Details

In general error return codes are used to indicate incorrect usage rather than raised syntax conditions. However, syntax errors are raised if the *methodName* argument is missing or the empty string, or invoking this method before the *underlying* dialog is created.

Raises syntax errors when some incorrect usage is detected.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if any F Key key press events happen.

Example:

The following example is a variation on the *example* shown for the *connectKeyPress* method. It connects all the function keys to the same method and then determines what action to take by examining which key was pressed.

```

::method initDialog

  ...

  -- Capture all function key presses.
  self~connectFKeyPress(onFKey)

  ...

::method onFKey unguarded
  use arg keyPressed

  select
    when keyPressed == .VK~F2 then self~showCustomerLookupDialog

    when keyPressed = 114 then do
```

```
        prodNum = self~newEdit(IDC_EDIT_PRODUCT)~getText
        if prodNum \== "" then self~showProductInfo(prodNum)
      end

      when keyPressed = 115 then self~resetAllFields
      when keyPressed = 116 then self~printInvoice

      otherwise do
        -- Not interested in any other function keys
        nop
      end
    end

    return 0
```

4.7.15.2. connectKeyPress

```
>>--connectKeyPress(--methodName--,--keys-+------------+--)-----><
                                          +-,--filter--+
```

The *connectKeyPress* method connects a key press *event* notification with a method in the Rexx dialog. A single key or multiple keys can be connected to the same method. Multiple methods can be connected for key press events, but only 1 method can be connected to any single key.

Arguments:

The arguments are:

methodName [required]

The name of the method that is to be invoked when the key press event happens. This argument can not be the empty string.

keys [required]

The key (or keys) for which the key press event is to be connected. A single key or multiple keys can be specified. A range of keys can be used. Each single key or range of keys is separated by a comma. A range of keys is denoted by using the dash character "-". White space within the *keys* argument is ignored. This argument can not be the empty string.

The keys are specified by the numeric value defined by Microsoft for its virtual key set. These numeric values are 0 through 255. There are some integer values between 0 and 255 that do not have a virtual key assigned to them. For example, 0, 7, 10, 11, and 255 are not used. The *VK* class contains constants for all of the defined virtual keys.

In addition, there are a few keywords that can be used to specify some common key ranges. These keywords are:

ALL

All keys.

FKEYS

All Function keys, other than F1. (In Windows the F1 key is the help key and the *connectHelp* method should be used for F1.)

ALPHA

The keys A though Z.

NUMERIC

The keys 0 through 9. Note that these are the normal number keys, not the keypad numbers on an enhanced keyboard.

ALPHANUMERIC

The keys A through Z and 0 through 9.

Note that case is insignificant for these keywords as is the order of the keywords. A keyword not in the list will result in a return of -9. However, if the argument contains other valid key tokens, those keys will be connected to the method. If there are no other valid key tokens, then no connection is made.

filter [optional]

A (simplistic) filter that is applied to the key press event for the key(s) specified. The filter is a string of keywords separated by blanks. (Case is not significant, neither is the order of the words. Any words other than the specified keywords are ignored.) The possible keywords are: **`SHIFT, CONTROL, ALT, AND, NONE, VIRTUAL.`**

The VIRTUAL keyword can be abbreviated to VIRT if desired. The VIRTUAL keyword effects how the test for the shift, control, and alt key is performed. By default the physical state of the keyboard is checked to see if the control, alt, or shift key is depressed. However, it is common in Windows to use keystroke programs that inject keystrokes into other application windows. Testing the physical state of the keyboard will not detect combination keystrokes like Ctrl-S, Alt-L, etc., that are inserted by keystroke programs because the physical state of the modifier keys control and alt will not be depressed. If the VIRTUAL keyword is used, the test for the modifier keys being down will be altered in a way that will detect if the virtual state of the key is down. This test will detect key events inserted into the Rexx application by third party keystroke programs.

Shift, control, and alt specify that the corresponding key must be down at the time of the key press event. These keywords are combined in a boolean expression. The default is an OR expression. If the AND keyword is present then the boolean expression is an AND expression. If the NONE keyword is used, it means that none of the shift, control, or alt keys can be down at the time of the key press event. (When NONE is used, all other words, except VIRTUAL, in the string are ignored.)

Some examples may make this more clear:

```
::method initDialog

  -- Using the below, the onAltCD method would be invoked when the user types
  -- Alt-Shift-C or Alt-Shift-D.  But the method would not be invoked for Alt-C
  -- or Shift-D (or any other key press event.)

  keys = .VK~C "," .VK~D
  self~connectKeyPress(onAltCD, keys, "ALT AND SHIFT")

  -- The below would invoke the onAltCD method any time a C or a D was typed
  -- with either the Alt or the Control key down.  This would include Alt-C,
  -- Alt-Shift-C, Ctrl-Alt-Shift-C, etc..

  self~connectKeyPress(onAltCD, keys, "ALT CONTROL")

  -- The below would invoke the onAltCD method only when Alt-C or Alt-D was
  -- typed.

  self~connectKeyPress(onAltCD, keys, "ALT AND")

  -- The below would invoke the onF4 method only when the F4 key was pressed by
  -- itself. Alt-F4, Ctrl-F4, etc., would not invoke the method.

  self~connectKeyPress(onF4, .VK~F4, "NONE")
```

Return value:

The possible return values are:

0

Success.

-2

The underlying mechanism in the Windows API that is used to capture key events failed.

-6

The maximum number of connections has been reached.

-7

The *methodName* method is already connected to a key down event for this dialog.

-8

The **`filter`** argument is not correct.

-9

An incorrect format for the **keys**. Note that it is possible to get a return of -9 but still have some keys connected. For instance in the following example the C and D keys would be connected and the filter applied. The ""dog"" token would result in -9 being returned:

```
keys = .VK~C ", dog," .VK~D
ret = self~connectKeyPress('onAltCD', keys, "ALT AND SHIFT")
say 'Got a return of:' ret
say "Have connection to onAltCD?" self~hasKeyPressConnection('onAltCD')

-- The output would be:
Got a return of: -1
Have connection to onAltCD? 1
```

Remarks

Unlike most other methods that connect event notifications, the underlying Windows dialog must exist before this method can be used. That means it can be used in *initDialog* or any time thereafter. There is a maximum limit of 63 methods, per dialog, that can be connected to key press events. Connections can be removed using the *disconnectKeyPress* method if there is no longer a need for a notification of a key press.

The dialog control object also has a *connectKeyPress*() method. It is important to note this distinction between the two methods. The method of the dialog object (this method) will capture all key press events when the dialog is the active window. This includes key presses when a dialog control in the dialog has the focus.

The method of the dialog control object will only capture key press events when the specific dialog control has the focus. This implies that if you connect the same key press event to both the dialog and to a specific dialog control, if the key press event occurs when the dialog control has the focus, you will receive two event notifications.

Details

In general error return codes are used to indicate incorrect usage rather than raised syntax conditions. However, syntax errors are raised for missing required arguments, using the empty string for required arguments, or invoking this method before the *underlying* dialog is created.

Raises syntax errors when some incorrect usage is detected.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if any draw events happen.

Due to the nature of key press events, the low-level implementation of capturing the key strokes is different from most of the other methods of the **`EventNotification`** class. There is no single message sent to the underlying dialog for a key stroke event.

Example:

The following example is from a fictitious customer order system. As the user is filling out a customer order using the customer order dialog, he has the F2 through F5 short cut keys available. F2 brings up a customer look up dialog. F3 looks up info on the product number entered in an edit control. F4 resets the form by clearing all the fields. F5 is used to print out the finished invoice.

```
::method initDialog

  ...

  -- Capture F2 key presses, but not Ctrl-F2 or Alt-F2, etc..
  self~connectKeyPress(onF2, .VK~VK_F2, "NONE")

  -- Same idea for F3, F4, and F5.  This uses the actual numeric value for the
  -- keys without bothering to use the VK class to translate.
  self~connectKeyPress(onF3, 114, "NONE")
  self~connectKeyPress(onF4, 115, "NONE")
  self~connectKeyPress(onF5, 116, "NONE")

  ...

::method onF2 unguarded
  self~showCustomerLookupDialog
  return 0

::method onF3 unguarded

  prodNum = self~newEdit(IDC_EDIT_PRODUCT)~getText
  if prodNum \== "" then self~showProductInfo(prodNum)
  return 0

::method onF4 unguarded
  self~resetAllFields
  return 0

::method onF5 unguarded
  self~printInvoice
  return 0
```

4.7.15.3. disconnectKeyPress

```
>>--disconnectKeyPress(--+--------------+--)-----><
                         +--methodName--+
```

The *disconnectKeyPress* method disconnects a key press event from a method that was previously connected using *connectKeyPress*, or *connectFKeyPress*.

Arguments:

The single argument is:

methodName [optional]

If *methodName* is specified, only the key press events connected to that method are disconnected. If the argument is omitted, then all key press events for the dialog will be disconnected.

Return value:

The possible return values are:

0

Success.

-2

While trying to disconnect the method, the underlying mechanism in the Windows API that is used to capture key events had an error. This is unlikely to happen.

-7

Either the *methodName* method is already disconnected, or there are no methods connected at all.

Remarks:

The dialog control object also has a *disconnectKeyPress* method. The method of the dialog object (this method) can only disconnect key press events that were set with the dialog object's versions of *connectKeyPress* and *connectFKeyPress* methods. This method can not disconnect key press events that were set with the dialog control object's versions of *connectKeyPress* and *connectFKeyPress* methods.

Details

Raises syntax errors when some incorrect usage is detected.

Example:

The following example is a variation on the *example* shown for the *connectKeyPress* method. It builds on the fictitious customer order system. The F7 key saves the completed invoice into the system and enters a different phase of the companies business process. At this point (for whatever fictitious business reason) the fields can no longer be cleared and the user is not allowed to look up customer or product information. But, the user may still need to print the invoice. To prevent the accidental press of the hot keys causing the wrong action, those key presses are disconnected.

To demonstrate how key press connections can be added and removed through out the life time of the dialog, this example adds the F9 hot key. F9 starts a new order entry cycle and re-connects the hot keys used during the creation of a customer invoice. When the user then saves the next completed invoice, key press connections are removed, when she starts a new invoice key press connections are restored. This cycle could continue though out the day without the user ever closing the main dialog.

```
::method initDialog

  ...

  -- Capture F2 key presses, but not Ctrl-F2 or Alt-F2, etc..
  self~connectKeyPress(onF2, .VK~F2, "NONE")

  -- Same idea for F3, F4, F5, and F7.  This uses the actual numeric value for
  -- the keys without bothering to use the .VK class to translate.
  self~connectKeyPress(onF3, 114, "NONE")
  self~connectKeyPress(onF4, 115, "NONE")
  self~connectKeyPress(onF5, 116, "NONE")
  self~connectKeyPress(onF7, 118, "NONE")
  self~connectKeyPress(onF9, 120, "NONE")

  ...

::method onF2 unguarded
  self~showCustomerLookupDialog
  return 0

::method onF3 unguarded

  prodNum = self~newEdit(IDC_EDIT_PRODUCT)~getText
  if prodNum \== "" then self~showProductInfo(prodNum)
  return 0

::method onF4
  self~resetAllFields
  return 0

::method onF5
  self~printInvoice
  return 0

::method onF7

  self~saveToDataBase
  self~disconnectKeyPress(onF2)
  self~disconnectKeyPress(onF3)
  self~disconnectKeyPress(onF4)
  return 0

::method onF9

  self~resetAllFields
  self~connectKeyPress(onF2, 112, "NONE")
  self~connectKeyPress(onF3, 114, "NONE")
  self~connectKeyPress(onF4, 115, "NONE")
  return 0
```

4.7.15.4. hasKeyPressConnection

```
>>--hasKeyPressConnection(--+--------------+--)--><
                            +--methodName--+
```

This method is used to query if a connection to a key press event already exists.

Arguments:

The single optional argument is:

methodName [optional]

Query if any key press events are connected to the specified method. If this argument is omitted, the query is if any key press events are connected to **any** methods.

Return value:

Returns `.true` if the method is connected to a key press event or `.false` otherwise.

Details

The dialog control object also has a *hasKeyPressConnection* method. The method of the dialog object (this method) can only check for connections that were set with the dialog object's versions of *connectKeyPress* and *connectFKeyPress* methods. This method can not check for connections that were set with the dialog control object's versions of *connectKeyPress* and *connectFKeyPress* methods.

Example:

The following example could come from a dialog where the user has the option to use hot keys or not. When the reset button is pushed the state of the dialog fields are reset. The hot keys enabled check box is set to reflect whether hot keys are currently enabled or not.

```
::method defineDialog

  ...
  self~createCheckBox(IDC_CHECK_FKEYSENABlED, 30, 60, , , , "Hot Keys Enabled")
  ...
  self~createPushButton(IDC_PB_RESET, 60, 135, 45, 15, , "Reset", onReset)
  ...

::method onReset unguarded

  ...
  if self~hasKeyPressConnection then
    self~newCheckBox(IDC_CHECK_FKEYSENABlED)~check
  else
    self~newCheckBox(IDC_CHECK_FKEYSENABlED)~uncheck
  ...
  return 0
```

4.7.15.5. KeyPress Event Handler

The event handler for the key press event is invoked when a key connected through the *connectKeyPress* or the *connectFKeyPress* method is typed when the dialog has the focus.

The interpreter replies immediately to the operating system when the key is pressed. The event handler is then invoked concurrently as a separate activity and the interpreter does not wait for the return. The value returned from the event handler is ignored and the programmer need not return a value from the handler. However, good practice would be to always return a value from an event handler.

```
::method onKeyPress unguarded
  use arg keyCode, shift, control, alt, extraInfo

  return 0
```

Arguments:

The event handling method receives 5 arguments:

keyCode
: The numeric code of the key pressed.

shift
: A boolean (true or false) that denotes whether a shift key was down or up at the time of the key press. It will be true if a shift key was down and false if the shift key was not down.

control
: True if a control key was down at the time of the key press, false if it was not.

alt
: True if an alt key was down at the time of the key press, false if it was not.

extraInfo

This argument is a string containing keywords. It supplies extra information about the keyboard state at the time of a key press event. The string will contain some combination of these keywords

numOn

Num Lock was on at the time of the key press event.

numOff

Num Lock was off.

capsOn

Caps Lock was on at the time of the key press event.

capsOff

Caps Lock was off.

scrollOn

Scroll Lock was on at the time of the key press event.

scrollOff

Scroll Lock was off.

lShift

The left shift key was down at the time of the key press event.

rShift

The right shift key was down.

lControl

The left control key was down at the time of the key press event.

rControl

The right control key was down.

lAlt

The left alt key was down at the time of the key press event.

rAlt

The right alt key was down.

Return:

The return value from the event handler is discarded. Zero would make a good value to return.

Example

The following example comes from an application where the user can type F3 to close the current dialog and Alt-F3 to close the entire application. The event handler checks that only the Alt key is down when the F3 key is pressed and closes the application if that is true. Otherwise, if only the F3 key is pressed, then the handler closes the dialog:

```
::method initDialog
  self~connectKeyPress(onQuitKey, .VK~F3)

::method onQuitKey
  use arg key, shift, control, alt, info
  say 'Key press:' key 'shift?' shift 'control?' control || -
      ' alt?' alt 'extra info:' info

```

```
   if shift then return
   if control then return
   if alt then return self~quitApplication

   -- Only F3 is pressed, close this dialog
   return self~cancel:super
```

4.7.16. connectListBoxEvent

```
>>--connectListBoxEvent(--id--,--event--+----------+--+-------------+--)------><
                                        +-,--mName-+  +-,-willReply--+
```

Connects a method in the Rexx dialog to the Windows *event* notification from a *ListBox* control.

Arguments:

The arguments are:

id [required]
: The resource ID of the list box whose event notification is to be connected to the method.

event [required]
: A single keyword, case is not significant, which specifies the event to be connected:

  DBLCLK
  : An entry in the list box has been selected with a double click.

  ERRSPACE
  : An out-of-memory problem has occurred.

  GOTFOCUS
  : The list box has gained the input focus.

  LOSTFOCUS
  : The list box has lost the input focus.

  SELCANCEL
  : The selection in the list box has been canceled.

  SELCHANGE
  : Another list box entry has been selected.

mName [optional]
: The name of the method to be invoked whenever the specified notification is received from the list box. Provide an event handling method with a matching name. If this argument is omitted, ooDialog generates a name that consists of the event keyword preceded by **on**. For example *onLostFocus*.

willReply [optional]
: The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.false`**.

Return value:

The return codes are:

0
: No error detected.

-1
: The resource ID could not be resolved or the event argument is incorrect.

1
: The message was not connected correctly.

Remarks:

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

Example:

The following example displays the text of the selected list box entry each time the user selects a new item:

```
::class MyDlgClass subclass UserDialog

::method init
  self~connectListBoxEvent(IDC_LB_PLAYLIST, "SELCHANGE", "onSelectionChanged")

::method onSelectionChanged unguarded
  use arg , , id, code, lbox

  say "New selection is:" lbox~selected
  return 0
```

4.7.16.1. General ListBox Event Handler

The event handler for all of the list box events receives the same arguments and is coded in the same fashion. When the handler is invoked is fairly clear from the name of the event. The GOTFOCUS event handler is invoked when the list box gets the focus, etc..

The *willReply* argument in the *connectListBoxEvent* method determines how the event handler needs to respond to the notification.

```
::method onListBoxEvent unguarded
  use arg info, hwnd, id, notifyCode, listBox

  return 0
```

Arguments:

The event handling method receives 5 arguments. The first and second arguments need to be retained for backwards compatibility, but only the last 3 arguments are really needed:

info
: A numeric value that contains info about the event. The low order word contains the resource ID of the list box sending the event. The high order word contains the list box event code. The *loWord* and *hiWord* methods of the *DlgUtil* class can be used to extract these values. However, the ooDialog framework now extracts those values for you and sends them as the third and fourth arguments.

hwnd
: The window handle of the list box that sent the notification.

id
: The numeric resource id of the list box sending the notification.

notifyCode
: The numeric notification code of the event that caused the notification to be sent. Each dialog control has its own specific notification codes.

listBox
: The Rexx list box object that represents the underlying dialog control that sent the notification. It is possible this will be the **`.nil`** object if some error happened, but this is very unlikely.

Return:

The operating system ignores the return from this notification. 0 makes a good return value.

Example

The following example comes from an application that fills a list box with the names of movies. When the user clicks on a movie in the list box, the application fills a number of other controls with information specific to the selected movie:

```
::method onMovieClick unguarded
   expose movieData
   use arg , , id, nCode, listBox

   movieName = listBox~selected

   combo = self~newComboBox(35)
   combo~deleteAll

   -- If no movie in the list box is selected, set everything blank, otherwise,
   -- set everything to match the selected movie.
   if movieName == "" then do
     self~newEdit(32)~setText("")
     self~newEdit(33)~setText("")
     self~newEdit(34)~setText("")
     combo~add("")
   end
   else do
     d = movieData[movieName]
     self~newEdit(32)~setText(d~produced)
     self~newEdit(33)~setText(d~star)
     self~newEdit(34)~setText(d~director)

     do n over d~with
        combo~add(n)
     end
     combo~selectIndex(1)
   end

   return 0
```

4.7.17. connectListViewEvent

```
>>-connectListViewEvent(--id--,--event--+---------------+--+-------------+--)--><
                                        +-,--methodName-+  +-,-willReply-+
```

The *connectListViewEvent* method connects a particular *event* notification from a list-view control with an event handling method in the Rexx dialog.

Arguments:

The arguments are:

id [required]

The ID of the list-view control for which a notification is to be connected. This can be symbolic or numeric.

event [required]

The event keyword. Use exactly one of the following keywords, case is not significant. Each keyword in the list is linked to the event handler documentation for that event:

ACTIVATE	CLICK	FOCUSCHANGED
BEGINDRAG	COLUMNCLICK	GETINFOTIP
BEGINEDIT	DBLCLK	INSERTED
BEGINRDRAG	DEFAULTEDIT	KEYDOWN
BEGINSCROLL	DELETE	KEYDOWNEX
CHANGING	DELETEALL	SELECTCHANGED
CHANGED	ENDEDIT	SELECTFOCUSCHANGED
CHECKBOXCHANGED	ENDSCROLL	

ACTIVATE

A list view item has been activated. One way an item is activated by is double-clicking the item with the left mouse button.

BEGINDRAG

A drag-and-drop operation was initiated. See *defListDragHandler* for information on how to implement a drag-and-drop handler.

BEGINEDIT

Editing a label has been started. Do not connect this event if you are using the DEFAULTEDIT keyword. The results are undefined. The list-view must have the *EDIT* style for this notification to be sent.

The event notification for this event has been enhanced since the original ooDialog implementation. To use the enhanced event notification, the *willReply* argument must be used. The value of the argument, true or false, does not matter. If *willReply* is omitted, the old style notification is used. The documentation for the *BEGINEDIT* event handler explains the difference between the two types of notifications.

BEGINRDRAG

A drag-and-drop operation involving the right mouse button was initiated. See *defListDragHandler* for information on how to implement a drag-and-drop handler.

BEGINSCROLL

A scrolling operation is starting.

CHANGED

An item has changed. The notification for this event is sent after the item changed.

CHANGING

An item is about to change. The notification for this event is sent before the item is changed.

CHECKBOXCHANGED

The check box state of an item changed. (The check box was checked or unchecked.) This event can only occur if the list-view has the check box *extended* list-view style. Use this keyword instead of the CHANGED keyword.

CLICK

This event notification is generated when the list-view is clicked with the left mouse button. However, in report view only, this excludes the column headers. Connecting the CLICK event is a replacement for the *connectNotifyEvent* method's CLICK event.

COLUMNCLICK

In report view only, a column header has been clicked. Contrast this with the CLICK keyword.

DBLCLK

This event is generated when the list-view is double-clicked with the left mouse button. However, in report view only, this excludes the column headers. Connecting the DBLCLK event is a replacement for the *connectNotifyEvent* method's DBLCLK event.

DELETE

An item has been deleted.

DELETEALL

All items have been deleted.

DEFAULTEDIT

This keyword is used to activate the internal handling of the list-view label editing operation. With this keyword, the ooDialog framework internally handles the BEGINEDIT and ENDEDIT notifications. The list-view must have the *EDIT* style for the BEGINEDIT / ENDEDIT notification to be sent. While using the DEFAULTEDIT connection may seem easier than using the BEGINEDIT / ENDEDIT connections, it does not provide the same flexibility as using the BEGINEDIT / ENDEDIT connections.

When you specify this event connection, omit the *methodName* argument, the arugment is ignored. Do not connect either the BEGINEDIT or ENDEDIT events when also using the DEFAULTEDIT connection. The result is undefined.

ENDEDIT

Label editing has ended. Do not connect this event if you are using the DEFAULTEDIT keyword. The results are undefined. The list-view must have the *EDIT* style for this notification to be sent.

The event notification for this event has been enhanced since the original ooDialog implementation. To use the enhanced event notification, the *willReply* argument must be used. The value of the argument, true or false, does not matter. If *willReply* is omitted, the old style notification is used. The documentation for the *ENDEDIT* event handler explains the difference between the two types of notifications.

ENDSCROLL

A scrolling operation has ended.

FOCUSCHANGED

The focus state of an item changed. (The item gained or lost the focus.) Use this keyword instead of the CHANGED keyword.

GETINFOTIP

The list-view control is requesting text to display an info tip. The notification is only sent when the list-view control has the extended *INFOTIP* style. The extended list-view styles must be set using the *addExtendedStyle* method.

INSERTED
A new item has been inserted.

KEYDOWN
A key was pressed inside the list view. This notification is not sent while a label is being edited.

KEYDOWNEX
A key was pressed inside the list view. This notification is not sent while a label is being edited.

This event is exactly the same as the KEYDOWN event. Except, when this keyword is used, the ooDialog framework sends a different set of arguments to the event handler. The additional arguments provide more information to the programmer than the arguments sent when the KEYDOWN keyword is used. The two keywords are needed to provide backwards compatibility.

SELECTCHANGED
The selection state of an item changed. (The item was selected or unselected.) Use this keyword instead of the CHANGED keyword.

SELECTFOCUSCHANGED
The selection state or the focus state of an item changed. This event argument combines the selection changed and the focus changed event into one connection. When this event is connected, separate selection changed and focus changed events can not be connected. This keyword can be abbreviated to SELECTFOCUS. Use this keyword instead of the CHANGED keyword.

methodName [optional]
The name of the event handling method. This method is invoked each time the specified event occurs for the list view control. The method name can not be the empty string. If you omit this argument, the event handler method name is generated for you. This name will be the event keyword, preceded by **on**. For example: *onColumnClick*.

willReply [optional]
The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is `.false`.

Note: The *willReply* argument is ignored for the GETINFOTIP event, the interpreter always waits for the reply. It makes no sense to connect this event when not planning to return a value.

Return value:

0
No error detected.

-1
The resource ID could not be resolved or the event argument is incorrect.

1
The event was not connected correctly. The error is likely caused by the message table being full, but could also indicate the interpreter is out of memory.

Remarks:

Microsoft continually enhances the Windows User Interface and therefore the dialog controls evolve over time. The **`connectListViewEvent`** method uses several event keywords that provide more information, for the same event, than that provided by the original ooDialog implementation. For instance, the CHECKBOXCHANGED, SELECTIONCHANGED, FOCUSCHANGED, and SELECTFOCUSCHANGED keywords all connect the same event as the CHANGED keyword. However, these keyword connections all provide more specific, detailed information in the arguments passed to the connected method than that provided by using the CHANGED keyword.

Likewise, the **`connectListViewEvent`** CLICK keyword provides much better information than that provided by the *connectNotifyEvent*'s CLICK keyword.

Note: If the same event, for the same control, is connected using two different connectXXX methods, only one connection will be in effect. Which connectXXX method is invoked is undefined. For example, take a dialog that has a list-view control with resource ID of 109. If the mouse click event is connected for that control using the **`connectNotifyEvent`** method and then the mouse click event is also connected using the **`connectListViewEvent`** method, only one connection will be active. Which one is active is undefined.

When using **`connectListViewEvent`** a separate method can be connected to each of the CHECKBOXCHANGED, SELECTIONCHANGED, and FOCUSCHANGED events. These event connections are all replacements for the CHANGED event.

Example:

The following example connects the column-clicked event for the list-view EMPLOYEES with method ColumnAction and changes the style of the list-view from REPORT to SMALLICON:

```
::class MyDlgClass subclass UserDialog

::method init
  self~init:super(...)
  self~connectListViewEvent("EMPLOYEES", "COLUMNCLICK", "columnAction")

::method columnAction unguarded
  use arg id, column
  lc = self~newListView("EMPLOYEES")
  lc~replaceStyle("REPORT", "SMALLICON EDIT SINGLESEL ASCENDING")
  if column > 0 then ...

  return 0
```

4.7.17.1. BeginDrag / BeginRDrag Event Handler

The event handler for the BegingDrag or BeginRDrag events is invoked when a drag-and-drop operation involving the left mouse button or the right mouse button is being initiated.

The *willReply* argument in the *connectListViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onBeginDrag unguarded
  use arg id, item, point, isLMB, listView

  return 0
```

Arguments:

The event handling method receives 5 arguments:

id

The numeric resource ID of the list view sending the notification.

item

The index of the item being dragged.

point

The point where the mouse cursor was pressed (x and y positions, separated by a blank).

isLMB

True if the drag was started with the left mouse button, false if the drag was started with the right mouse button.

listView

The Rexx list view object that represents the underlying list view that sent the notification.

Return:

The operating system ignores the return value from the event handler. 0 makes a good return value.

Example

The defListDragHandler() method is an example of coding a begin drag handler. This method can be found in the ooDialog.cls file.

4.7.17.2. BeginEdit Event Handler

The event handler for the BEGINEDIT event is invoked when the user begins a label editing operation on an item of the list-view. When label editing begins, an edit control is created by the operating system, but not positioned or displayed. Before it is displayed, the tree-view control sends a BEGINEDIT notification message. A label editing operation is only available when the list-view has the *EDIT* style.

In general, the programmer would connect both the BEGINEDIT and *ENDEDIT* notifications. Both of these event notifications have been enhanced since the original ooDialog implementation. If the *willReply* argument to the *connectListViewEvent* method is omitted the old implementation is used. If the argument is not omitted, the new implementation is used. How the two event handlers work is described separately.

New event handler description:

Whether the programmer must return a value and if the interpreter waits, or does not wait, for this return is determined by the value of the *willReply* argument. If *willReply* is true, the programmer must return true or false from the event handler and the interpreter waits for that reply. If *willReply* is false the interpreter does not wait for a reply.

```
::method onBeginEdit unguarded
  use arg id, itemID, editCtrl, listViewCtrl

  return zz
```

Arguments:

The event handling method receives 4 arguments:

id
: The resource id of the list-view sending the notification.

itemID
: The item index whose label is about to be edited.

editCtrl
: The Rexx edit control object used for the editing operation. The programmer can customize the editing operation by using the methods of the *Edit* class.

  Note that this object is only valid during the editing operation. When the user ends the editing, the operating system destroys the underlying edit control and the Rexx object is no longer valid. Each time the user starts a new editing operation, the operating system creates a new edit control.

listViewCtrl
: The Rexx list-view object whose underlying list-view control has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newListView* method. Unlike the *editCtrl* object, this object is valid as long as the dialog is executing.

Return:

When the programmer used true for the *willReply* argument, the event handler must return true or false. To allow the editing operation to begin, return true. To cancel the editing operation, return false. Returning a value from the event handler gives the programmer the option determining if the label for the specific list-view item should or should not be edited.

Example

The following example shows a possible event handler for the BEGINEDIT event. It uses the *itemIndex* argument to determine which item is about the have its label edited, and checks that editing is allowed for that item. If it is, it returns true to allow the operation. If it is not, it returns false to cancel the operation and puts up a message box to inform the user:

```
::method onBeginEdit unguarded
  use arg id, itemIndex, editCtrl, listViewCtrl

  rec = listViewCtrl~getItemData(itemIndex)
  if rec~isEditable then return .true

  reply .false

  msg = "The record for" rec~FirstName rec~LastName 'can not be changed.'
  title = "Label Edit Error"
  j = MessageDialog(msg, self~hwnd, title, , "WARNING")

  return
```

Old event handler description:

The old style event notification is used when the programmer omits the *willReply* argument in the invocation of the *connectListViewEvent* method. The return from the event handler is completely ignored, the interpreter does not wait for this return.

```
::method onBeginEdit unguarded
  use arg id, useless
```

Arguments:

The event handling method receives 2 arguments:

id

The resource id of the list-view sending the notification.

useless

This is an operating system value that has no meaning within Rexx code. It can not be used for anything and its value does not correlate with anything accessible to the Rexx programmer.

Return:

Returning, or not returning, a value has no meaning.

Remarks

Connecting this event and not using the *willReply* argument does not make much sense. The event handler really serves no purpose.

4.7.17.3. BeginScroll / EndScroll Event Handler

The event handler for the begin scroll event is invoked when a scrolling operation starts and the event handler for the end scroll event is invoked when a scrolling operation ends. The scrolling operation can be either horizontal or vertical scrolling. Both event handlers receive the same arguments and are basically coded the same.

The *willReply* argument in the *connectListViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onBeginScroll unguarded
  use arg ctrlID, dx, dy, listView, isBeginScroll

  return 0
```

Arguments:

The event handling method receives 5 arguments:

ctrlID

The resource ID of the list-view that generated the event notification.

dx

The Microsoft documentation says that this value specifies in pixels the horizontal position where a scrolling operation should begin. However, a similar description for the *dy* value is not correct, making it difficult to see if the Microsoft documentation is accurate for the *dx* value.

The value is 0 if the scrolling operation was vertical. It is negative if the horizontal scrolling was to the left, and positive if the operation was to the right.

dy

The Microsoft documentation says that this value specifies in pixels the vertical position where a scrolling operation should begin. However, experimentation shows that is actually the number of items scrolled.

The value is 0 if the scrolling operation was horizontal. It is negative if the vertical scrolling was towards the top, and positive if the operation was towards the bottom.

listView
: The Rexx list view object that represents the underlying list view that generated the event notification.

isBeginScroll
: The event handlers for both the BEGINSCROLL and ENDSCROLL are identical. This argument allows the Rexx programmer to connect both events to the same method and then distinguish in the event handler which event occurred. This argument is true if the event which caused the event handler to be invoked was the BEGINSCROLL event and false if the event was ENDSCROLL.

Return:

The value of the return is ignored by the operating system. 0 makes a good return value.

Example

The following example connects both the BEGINSCROLL and ENDSCROLL events to the same event handling method. The method simply prints out the values sent by the operating system:

```
::method onScroll unguarded
  use arg id, dx, dy, listView, isBeginScroll

  if isBegin then op = 'beginScroll'
  else op = 'endScroll'

  say 'id:' id 'dx:' dx 'dy:' dy 'listView:' listView op

  return 0

/* Output might be for example: */

id 200 dx: 5 dy: 0 listView: a ListView beginScroll
id 200 dx: 5 dy: 0 listView: a ListView endScroll
id 200 dx: 5 dy: 0 listView: a ListView beginScroll
id 200 dx: 5 dy: 0 listView: a ListView endScroll
id 200 dx: 5 dy: 0 listView: a ListView beginScroll
id 200 dx: 5 dy: 0 listView: a ListView endScroll
id 200 dx: 158 dy: 0 listView: a ListView beginScroll
id 200 dx: 158 dy: 0 listView: a ListView endScroll
id 200 dx: -173 dy: 0 listView: a ListView beginScroll
id 200 dx: -173 dy: 0 listView: a ListView endScroll
id 200 dx: 0 dy: 1 listView: a ListView beginScroll
id 200 dx: 0 dy: 1 listView: a ListView endScroll
```

4.7.17.4. CheckBoxChanged Event Handler

The event handler for the checkbox changed event is invoked when the user checks or unchecks a checkbox in the **`ListView`** control.

The *willReply* argument in the *connectListViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onCheckboxChanged unguarded
  use arg id, itemIndex, state, listView

  return 0
```

Arguments:

The event handling method receives 4 arguments:

id
: The resource ID of the list view control sending the notification message.

itemIndex
: The index of the item whose checkbox was changed.

state
: This argument reports whether the check box was checked or unchecked. Its value will be either *CHECKED* or *UNCHECKED*.

listView
: The Rexx list view object that represents the underlying list view that generated the event notification.

Return:

The operating system ignores the return from this event handler. 0 makes a good return values.

Example

The following example is from an address book application. A list-view control is filled with the information from the address book, one item for each entry. The check box changed event is connected to the *onCheckboxChanged* method. The *onCheckboxChanged* method will receive 3 arguments: the resource ID of the control, the index of the item whose check box changed, and the changed state. If the user checks the check box, that entry is added to a mail merge being constructed. If the user unchecks the box, the entry is removed from the mail merge.

```
::class MailingListDlg subclass UserDialog

::method initDialog
  expose mailList

  ...
  mailList = self~newListView(IDC_LV_ADDRESSES)
  ...

  -- Since the methodName argument is omitted, ooDialog will construct a default
  -- name of 'onCheckboxChanged'
  self~connectListViewEvent(IDC_LV_ADDRESSES, "CHECKBOXCHANGED")
  ...

::method onCheckboxChanged unguarded
  use arg id, itemIndex, state, mailList

  if state == "CHECKED" then
    self~addToMailMerge(mailList, itemIndex)
  else
    self~removeFromMailMerge(mailList, itemIndex)

  return 0
```

4.7.17.5. Click / Double Click Event Handler

The event handling method for the CLICK event is invoked when the user clicks on the list-view with the left mouse. The event handler for the DBLCLK is invoked when the user double clicks on the list-view. This excludes the column headers in report view. Both event handlers receive the same arguments and are coded the same way.

The *willReply* argument in the *connectListViewEvent* method determines how the event handler needs to respond to the notification.

Note that the user can click on a list-view item, or on the background of the list view. When the click is on the background of the list-view then both the *itemIndex* and *columnIndex* will be -1. The method will receive four arguments:

```
  ::method onClick unguarded
    use arg id, itemIndex, columnIndex, keyState, listView

    return 0
```

Arguments:

The event handling method receives 5 arguments:

id

The resource ID of the list view control sending the notification message.

itemIndex

The index of the item that was clicked on.

columnIndex

The index of the column in the item that was clicked on. If not in report view, this will always be 0.

keyState

One or more keywords, separated by blanks, that report the shift, alt, control key state. It will either be the single key word: *NONE*, or any combination of *SHIFT CONTROL ALT*. The keywords report which of the keys were pressed at the time of the mouse click or double click.

listView

The Rexx list view object that represents the underlying list view that generated the event notification.

Return:

The operating system ignores the return from this event handler. 0 makes a good return values.

4.7.17.6. ColumnClick Event Handler

The event handler for the column click event is invoked when when the user clicks on a column header in a list view in report view.

The *willReply* argument in the *connectListViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onColumnClick unguarded
  use arg id, col, listView

  return 0
```

Arguments:

The event handling method receives 3 arguments:

id

The resource ID of the list view control sending the notification message.

col

The zero-based index of the column that was clicked.

listView
: The Rexx list view object that represents the underlying list view that generated the event notification.

Return:

The operating system ignores the return for this notification. 0 makes a good return value.

Example

The following example comes from and application that allows the user to sort the columns in the list view when it is in report view. When the application is first opened none of the columns are sorted and the first click on any column sorts the list-view items on that column in ascending order. Subsequent clicks on the column header reverses the current state. If the column is sorted in ascending order, a click sorts it in descending order ...

```
::method connectEvents private

    self~connectButtonEvent(IDC_RB_REPORT, "CLICKED", onReport, .true)
    self~connectButtonEvent(IDC_RB_LIST, "CLICKED", onList, sync)
    self~connectButtonEvent(IDC_RB_ICON, "CLICKED", onIcon)
    self~connectButtonEvent(IDC_RB_SMALL_ICON, "CLICKED", onSmallIcon, .false)
    self~connectListViewEvent(IDC_LV_VIEWS, "COLUMNCLICK", onColClick, .true)
    self~connectListViewEvent(IDC_LV_VIEWS, "BEGINDRAG", defListDragHandler)


::method onColClick unguarded
    expose itemColumns
    use arg id, colIndex, listView

    -- Adjust for 0-based indexes
    i = colIndex + 1

    d = .directory~new
    d~column = colIndex
    d~caseless = .false

    if itemColumns[i] == -1 then do
      d~ascending = .true
      itemColumns[i] = 0
    end
    else if itemColumns[i] == 0 then do
      d~ascending = .false
      itemColumns[i] = 1
    end
    else do
      d~ascending = .true
      itemColumns[i] = 0
    end

    listView~sortItems('InternalListViewSort', d)
    return 0
```

4.7.17.7. EndEdit Event Handler

The event handler for the ENDEDIT event is invoked when the user finishes a label editing operation on an item of the list-view. A label editing operation is only available when the list-view has the *EDIT* style.

In general, the programmer would connect both the *BEGINEDIT* and ENDEDIT notifications. Both of these event notifications have been enhanced since the original ooDialog implementation. If the *willReply* argument to the *connectListViewEvent* method is omitted the old implementation is used.

If the argument is not omitted, the new implementation is used. How the two event handlers work is described separately.

New event handler description:

The *willReply* argument in the *connectListViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onEndEdit unguarded
  use arg id, itemID, text, listViewCtrl

  return trueOrFalse
```

Arguments:

The event handling method receives 4 arguments:

id

The resource id of the list-view sending the notification.

itemID

The item index whose label being edited.

text

If the user canceled the edit operation then the *text* argument will be the .nil object. If the edit operation was not canceled then this argument will be the text the user entered.

listViewCtrl

The Rexx list-view object whose underlying list-view control has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newListView* method.

Return:

When the programmer used true for the *willReply* argument, the event handler must return true or false. To accept the edit text, return true. To disallow the change to the label, return false. If, the edit operation was canceled by the user, the operating system ignores the return from the event handler. Returning a value from the event handler gives the programmer the option of determining if the new label for the specific list-view item is acceptable.

Example

The following example checks the new text entered by the user. The label for the list-view items is displayed as last name, comma, first name. If the user's text is not in that format, the new text is rejected by returning false:

```
::method onEndEdit unguarded
use arg id, itemIndex, text, listViewCtrl

if text == .nil then return .false

if text~words == 2 & text~word(1)~right(1) == ',' then do
  reply .true

  rec = listViewCtrl~getItemData(itemIndex)
  rec~FirstName = text~word(2)
  rec~LastName  = text~word(1)~strip('T', ',')

  return
end

reply .false
```

```
msg = "The format for a record label must be" || .endOfLine || -
      "last name, comma, first name.  For"    || .endOfLine || -
      "example: Swift, Tom"                   || .endOfLine~copies(2) || -
      "The change is rejected."

title = "Label Editing Error"
j = MessageDialog(msg, self~hwnd, title, , "WARNING")

return
```

Old event handler description:

The old style event notification is used when the programmer omits the *willReply* argument in the invocation of the *connectListViewEvent* method. The return from the event handler is completely ignored, the interpreter does not wait for this return. If the user canceled the edit operation, the label will be unchanged. If the user did not cancel the edit operation, the label of the item is changed to the text the user entered.

```
::method onEndEdit unguarded
  use arg id, itemIndex, maybeText, listView
```

Arguments:

The event handling method receives 4 arguments:

id
: The resource id of the list-view sending the notification.

itemIndex
: The index of the list-view item that was edited.

text [optional]
: If the user canceled the edit operation, the *text* argument is omitted. If the user did not cancel, then the *text* argument is the text the user entered.

listView
: The Rexx list-view object that represents the underlying list-view control has sent the notification.

Return:

Returning, or not returning, a value has no meaning. The interpreter does not wait for the return and its value, if any is discarded. However, best practice would be to always return a value from an event handler, 0 makes a good return value.

Remarks

When the user does not cancel the edit operation, the operating system automatically changes the label of the item to what the user entered. To prevent this behavior, the programmer needs to use the new style event handler by using the *willReply* argument to the *connectListViewEvent* method.

4.7.17.8. FocusChanged Event Handler

The event handler for the focus changed event is invoked when a list view item gains or loses the focus.

The *willReply* argument in the *connectListViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onFocusChanged unguarded
  use arg id, itemIndex, state, listView

  return 0
```

Arguments:

The event handling method receives 4 arguments:

id
: The resource ID of the list view control that sent the notification.

itemIndex
: The index of the item which gained or lost the focus.

state
: This argument reports whether the focus was gained or lost. Its value will be either *FOCUSED* or *UNFOCUSED*.

listView
: The Rexx **`ListView`** object whose underlying Windows list view sent the notification.

Return:

The operating system ignores the return from this notification. 0 make a good return value.

4.7.17.9. General ListView Event Handler

A number of the list view events are processed internally by ooDialog in the same manner. The event handlers for these events are sent the same arguments and are essentially coded the same way. This *general* event handler is described here for the ACTIVATE, CHANGED, CHANGING, DELETE, DELETEALL, INSERTED events. The description for event keyword in the *connectListViewEvent* documentation details when each event will invoke this event handler.

The *willReply* argument in the *connectListViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onListViewEvent unguarded
  use arg id, ptr, notifyCode, listView

  return 0
```

Arguments:

The event handling method receives 4 arguments:

id
: The numeric resource ID of the list view sending the notification.

ptr
: The numeric value of a pointer to a memory location in the running application. This is of no conceivable use to the ooDialog programmer, but is retained for backwards compatibility.

notifyCode
: The numeric notification code of the event that caused the notification to be sent. Each dialog control has its own specific notification codes.

listView
: The Rexx list view object that represents the underlying dialog control that sent the notification. It is possible this will be the **`.nil`** object if some error happened, but this is very unlikely.

Return:

The implementation for these general list view events has been in ooDialog for a very long time. The implementation returns 0 to the operating system. 0 makes a good return value.

However, beginning in ooDialog 4.2.4, ***if*** the *willReply* argument in the *connectListViewEvent* method is set to true by the programmer, the ooDialog framework will take the value returned from the event handler and return that value to the operating system. See the remarks section for some guidance on the meaning of a returned value for these general events.

Remarks:

The return from the event notifications in this *general* category is ignored by the operating system for some of the events, but has meaning for some of the other events. ooDialog has been enhanced to allow the programmer to return a meaningful value to the operating system from this general event handler. If the *willReply* argument is set to true, the interpreter waits for the return value from the event handler and returns that value to the operating system. Here is a brief description of the meaning for the returned value for the general events:

ACTIVATE
: MSDN says the event handler must return 0 for this event.

CHANGED
: The operating system ignores the return for this event. 0 makes a good return value.

CHANGING
: Return true to prevent the change or false to allow the change. Note however that this general event handler is not sent sufficient information for the programmer ot make a good decision if she should prevent the change. A future enhancement to ooDialog may allow better handling of this event notification.

DELETE
: The operating system ignores the return for this event. 0 makes a good return value.

DELETEALL
: The operating system sends this notification when all items in a list view are about to be deleted. After that a DELETE notification is sent for each item that is deleted. If the event handler returns true for the DELETEALL notification the individual DELETE notifications are not sent. If the event handler returns false, the individual notifications are sent.

INSERTED
: The operating system ignores the return for this event. 0 makes a good return value.

4.7.17.10. GetInfoTip Event Handler

The event handler method connected to the get info tip event is invoked when the list-view control requests the text to display in the info tip. The programmer must return a string value and the interpreter waits for this return. The *willRepy* argument of the *connectListViewEvent* method is ignored for this event.

```
::method onGetInfoTip unguarded
```

```
  use arg id, itemIndex, text, maxLen, listView

  return infoText
```

Arguments:

The event handling method receives 5 arguments:

id
: The resource ID of the list-view control requesting the info tip text.

itemIndex
: The index of the list-view item that the info tip is for.

text
: The current text the list-view intends to display. Note that most often this is the empty string. However, in some cases it will not be the empty string. For instance, in report view, if the column is not wide enough to display the entire text for the item, the *text* argument will contain the entire item's text. Microsoft suggests that if *text* is not the empty string, the application should append its text to the end of the string.

maxLen
: The maximum length of the string that will be displayed. The programmer should not assume what this length is. However, testing shows that it is usually 1023. If the returned text is longer than the *maxLen* value, the text will automatically be truncated to *maxLen* characters.

listView
: The Rexx list view object that represents the underlying dialog control that sent the notification. It is possible this will be the **`.nil`** object if some error happened, but this is very unlikely.

Return:

The event handler must return a string value. If the string is not the empty string, it will be displayed as the info tip. If the empty string is returned, then the previous value of *text* is displayed. That is, if *text* is the empty string, no info tip will be shown. However, if *text* is not the empty string, that text will be displayed unchanged.

Example

The following example example displays an info tip that shows expanded record information for the list-view item. The *useInfoTips* variable is used to determine if an info tip should be displayed or not. If *useInfoTips* is false, no tip is displayed. If true the record information is formatted and returned. Note that new line characters are used to break up the information into lines:

```
::method onGetInfoTip unguarded
  expose records useInfoTips
  use arg id, item, text, maxLen, listView

  text = ''

  if useInfoTips then do
    r = records[item + 1]
    text = r~firstName r~lastName '('r~age')' || .endOfLine || -
           r~street                           || .endOfLine || -
           r~city',' r~state r~zipcode
  end

  return text
```

4.7.17.11. KeyDown Event Handler

The event handler for the key down event is invoked when the user types a key when the list view has the focus.

The *willReply* argument in the *connectListViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onKeyDown unguarded
  use arg id, vKey, listView

  return zz
```

Arguments:

The event handling method receives 3 arguments:

id

The resource ID of the list-view control that sent the notification.

vKey

The virtual key code of of the key pressed. The *VK* class can be used to determine which key was pressed.

listView

The Rexx list view object that represents the underlying dialog control that sent the notification. It is possible this will be the **`.nil`** object if some error happened, but this is very unlikely.

Return:

The operating system ignores the return for this notification. 0 make a good return value.

Example

The following example is from an application that allows the user to delete a list view item by pressing the Del key when the item to be deleted is selected. The list view is single selection

```
::method onKeyDown unguarded
  use arg id, vkey, listView

  if vKey == .VK~DELETE then do
    selectedIndex = listView~selected
    if selectedIndex <> -1 then do
      listView~delete(selectedIndex)
      if selectedIndex == listView~items then do
        if selectedIndex == 0 then do
          -- Do nothing, there are no items in the list-view now.
          nop
        end
        else do
          listView~select(selectedIndex - 1)
        end
      end
      else do
        listView~select(selectedIndex)
      end
    end
  end

  return 0
```

4.7.17.12. KeyDown (extended) Event Handler

The event-handling method connected through the KEYDOWNEX keyword is similar to the event handler for the *KeyDown* event handler. It is invoked for the same event, when the user presses a key within the list-view. However, it receives a different set of arguments than that provided when the KEYDOWN keyword is used. The arguments sent to the KEYDOWNEX event handler provide more information about the event. This allows the programmer to write a more flexible event handler.

The *willReply* argument in the *connectListViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onKeyDownEx unguarded
  use arg vKey, isShift, isCtrl, isAlt, extraInfo, listViewObj, id

  return response
```

Event Handler Method Arguments:

The event handling method receives 7 arguments:

vKey

The virtual key code of of the key pressed. The *VK* class can be used to determine which key was pressed.

isShift

A boolean (true or false) that denotes whether a shift key was down or up at the time of the key press. It will be true if a shift key was down and false if the shift key was not down.

isCtrl

True if a control key was down at the time of the key press, false if it was not.

isAlt

True if an alt key was down at the time of the key press, false if it was not.

extraInfo

This argument is a string containing keywords. It supplies extra information about the keyboard state at the time of a key press event. The string will contain some combination of these keywords

extended

The character event is for one of the extended keys previously mentioned, INS, DEL, HOME, END, PAGE UP, PAGE DOWN, or one of the arrow keys.

numOn

Num Lock was on at the time of the key press event.

numOff

Num Lock was off.

capsOn

Caps Lock was on at the time of the key press event.

capsOff

Caps Lock was off.

scrollOn

Scroll Lock was on at the time of the key press event.

scrollOff
: Scroll Lock was off.

lShift
: The left shift key was down at the time of the key press event.

rShift
: The right shift key was down.

lControl
: The left control key was down at the time of the key press event.

rControl
: The right control key was down.

lAlt
: The left alt key was down at the time of the key press event.

rAlt
: The right alt key was down.

listViewObj
: The Rexx **`ListView`** object whose underlying Windows list-view had the keydown event.

id
: The resource ID of the list-view control that sent the notification.

Return:

When the *willReply* argument to the *connectListViewEvent* method is true, the event handler must return a value. However, the operating system ignores the return value, so any value can be used. 0 makes a good return.

If *willReply* is false or SYNC, the event handler does not *have* to return a value, but good practice would be to always return a value from an event handler.

Example

The following example uses an event handler that deletes the selected item if the user presses the DEL key while holding down the control key. The next item following the deleted item is then selected. However, if the deleted item is the last item in the last, then the previous item is selected. If the deleted item is the last item in the list, then nothing is selected:

```
::method onKeyDownEx unguarded
    use arg vKey, isShift, isCtrl, isAlt, extraInfo, listViewObj, id

    if vKey == .VK~DELETE & isCtrl then do
        selectedIndex = listView~selected
        if selectedIndex <> -1 then do
            listView~delete(selectedIndex)
            if selectedIndex == listView~items then do
                if selectedIndex == 0 then do
                    -- Do nothing, there are no items in the list-view now.
                    nop
                end
                else do
                    listView~select(selectedIndex - 1)
                end
            end
            else do
                listView~select(selectedIndex)
            end
          end
      end

    return 0
```

4.7.17.13. SelectChanged Event Handler

The event handler for the select changed event is invoked when only the selection state of a list view item has changed.

The *willReply* argument in the *connectListViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onSelectChanged unguarded
  use arg id, itemIndex, state, listView

  return 0
```

Arguments:

The event handling method receives 4 arguments:

id
: The resource ID of the list view that sent the notification.

itemIndex
: The index of the list view item whose selection was changed.

state
: This argument reports whether the item was selected or unselected. Its value will be either *SELECTED* or *UNSELECTED*.

listView
: The Rexx **`ListView`** object that represents the underlying Windows list view that sent the notification.

Return:

The operating system ignores the return from this notification. 0 makes a good return value.

4.7.17.14. SelectFocusChanged Event Handler

The event handler for the selection or focus changed event is invoked when either the selection or the focus of a list view item changes.

The *willReply* argument in the *connectListViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onSelectFocusChanged unguarded
  use arg id, itemIndex, listView

  return 0
```

Arguments:

The event handling method receives 4 arguments:

id
: The resource ID of the list view that sent the notification.

itemIndex
: The index of the list view item where the state was changed.

state
: This argument reports whether the focus was gained or lost and whether the selection was gained or lost. Its value will contain at least one of the keywords: *SELECTED*, *UNSELECTED*, *FOCUS*, or *UNFOCUSED*. It is possible for both the selection and focus changed to be reported at once, however sometimes each change is reported separately. (This has nothing to do with ooDialog, it is how the operating system sends the messages.)

listView
: The Rexx **`ListView`** object that represents the underlying Windows list view that sent the notification.

Return:

The operating system ignores the return from this notification. 0 makes a good return value.

4.7.18. connectMonthCalendarEvent

```
>>--connectMonthCalendarEvent(--id-,-event-+---------+-+------------+--)-------><
                                           +-,-mName-+ +-,-wilReply-+
```

The *connectMonthCalendarEvent* method connects an *event* notification message from a *MonthCalendar* control to a method in the Rexx dialog.

Arguments:

The arguments are:

id [required]
: The resource ID of the month calendar control. May be numeric or *symbolic*.

event [required]
: Exactly one of the following keywords. The keyword specifies the event to be connected. Case is not significant:

GETDAYSTATE	SELCHANGE	VIEWCHANGE
RELEASED	SELECT	

GETDAYSTATE
: Sent by a month calendar control to request information about how individual days should be displayed. This notification message is only sent if the month calendar control has the DAYSTATE style. The *willReply* argument is ignored for this event, the event handler must *return* a reply.

Sent by the month calendar when the control is releasing the mouse capture. The return value from the event handler is ignored for this event.

SELECT
: Sent by a month calendar control when the user makes an explicit date selection within the control. The return value from the event handler is ignored for this event.

SELCHANGED

Sent by a month calendar control when the currently selected date or range of dates changes. The return value from the event handler is ignored for this event.

VIEWCHANGE

Requires Windows Vista or later. Sent by a month calendar control when the current view changes. The return value from the event handler is ignored for this event.

mName [optional]

The name of the method that is to be invoked whenever the specified notification is received from the month calendar control. The programmer defines this method. If this argument is omitted, a method name is automatically generated that consists of the event keyword preceded by **on**. For instance, **`onGetDayState`**. If the method name is supplied, it can not be the empty string.

willReply [optional]

The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.true`**.

However, this argument is ignored for the GETDAYSTATE event. If the programmer connects the GETDAYSTATE, the event handler must return a value. This can not be changed.

Return value:

This method returns **`.true`** if the event notification was connected correctly, otherwise **`.false`** .

Remarks:

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

Details:

Syntax errors are raised when incorrect usage is detected, including the use of an invalid symbolic ID or an unrecognized event keyword. A syntax error is raised if the programmer tries to connect the VIEWCHANGED event when the operating system is not Windows Vista or later.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if the connected event happens.

The underlying dialog receives the MCN_* messages as the notifications for the month calendar events.

Example:

The following example updates the text of a static control whenever the user selects a new date in the calendar.

```
::method initDialog

  self~connectMonthCalendarEvent(IDC_MC, "SELECT", onSelect)

::method onSelect unguarded
  expose dateText
  use arg startDate, endDate

  dateText~setText(self~formatDate(startDate))
  return 0
```

4.7.18.1. GetDayState Event Handler

```
::method onGetDayState unguarded
  use arg startDate, count, id, hwnd

  return dayStateBuffer
```

The event handler for the get day state event is invoked when the month calendar control requests information on how to display days in the calendar. The notification is only sent when the month calendar has the DAYSTATE style. The programmer must reply to this notification and **must** use the *DayStates* class to properly construct the reply. The interpreter waits for the reply.

The reply is a buffer containing a sequential collection of *DayState* values. Each individual day state value specifies how each day in a single month should be displayed. If a day in the day state value is turned on, the day is displayed in bold. If a day is not turned on, it is displayed with no emphasis. The **`DayStates`** and **`DayState`** classes provide methods to properly construct the day state values and the buffer containing the values.

Essentially, the programmer constructs a number of day state values and then returns a buffer containing those values.

Arguments:

The event handling method receives three arguments:

startDate

A **`DateTime`** object that specifies the start date the month calendar control needs day state values for. Each day state value specifies the state for every day in a month, even if the *dayState* arg is a date in the middle of a month. I.e., if the start date is January 11, 2011, the first day state value should be for the month of January.

count

The number of day state values required.

id

The resource ID for the month calendar control requesting the information.

hwnd

The window *handle* for the month calendar control requesting the information.

Return:

The reply is a buffer containing a sequential collection of *DayState* values. Each individual day state value specifies how each day in a single month should be displayed. If a day in the day state value is turned on, the day is displayed in bold. If a day is not turned on, it is displayed with no emphasis. The returned buffer must be constructed by using the *DayStates* class.

The **`DayStates`** and **`DayState`** classes provide methods to properly construct the day state values and the buffer containing the values. Essentially, the programmer constructs a number of day state values and then returns a buffer containing those values.

Example

The following example is used in a application that displays the 1st and the 15th of each month in bold. The start date can be ignored in this case because the day state value is the same for any month.

```
::method initDialog
```

```
  -- Connect the GETDAYSTATE event.
  self~connectMonthCalendarEvent(IDC_MC_PAYDAYS, "GETDAYSTATE", onGetDayState)

::method onGetDayState unguarded
  use arg startDate, count, id, hwnd

  dayStates = .array~new(count)
  do i = 1 to count
    dayStates[i] = .DayState~new(1, 15)
  end

  buffer = .DayStates~makeDayStateBuffer(dayStates)
  return buffer
```

4.7.18.2. Released Event Handler

```
::method onReleased unguarded
  use arg id, hwnd

  return 0
```

The event handler for the released event is invoked when the month calendar releases the mouse capture. The interpreter waits, or does not wait, for the reply as specified by the programmer in the *connectMonthCalendarEvent* method. The operating system ignores the value of the reply.

Arguments:

The event handling method receives two arguments:

id

The resource ID of the month calendar sending the notification.

hwnd

The window *handle* for the month calendar control sending the notification

Return:

Since the return value is ignored by the operating system, any value can be used. Typically, 0 is returned.

4.7.18.3. SelChanged Event Handler

```
::method onSelChanged unguarded
  use arg selStart, selEnd, id, hwnd

  return 0
```

The event handler for the selection changed event is invoked when the currently selected date or range of dates changes. This notification is sent when the user explicitly changes the selection within the current month or when the selection is implicitly changed by the user navigating to another month. The operating system also sends this notification at regular intervals so that the month calendar control can respond to date changes.

The notification is similar to the *SELECT* notification, except that the SELECT notification is only sent when the user explicitly changes the date. This notification is sent when the selected date is changed for any reason.

The interpreter waits, or does not wait, for the reply as specified by the programmer in the *connectMonthCalendarEvent* method. The operating system ignores the value of the reply.

Arguments:

The event handling method receives three arguments:

selStart

A **`DateTime`** object that is the new selected date, or the first selected date in a range of selected dates.

selEnd

A **`DateTime`** object that is the last selected date in a range of selected dates. If only a single date is selected, then *selEnd* will be the same date as *selStart*.

id

The resource ID of the month calendar sending the notification.

hwnd

The window *handle* for the month calendar control sending the notification.

Return:

The return value is ignored by the operating system and the programmer can return any value. Typically 0 is returned.

4.7.18.4. Select Event Handler

```
::method onSelect unguarded
  use arg selStart, selEnd, id, hwnd

  return 0
```

The SELECT event handler is invoked when the user explicitly selects a new date. Contrast this with the *SELCHANGED* event handler which is invoked when the selected date is changed for any reason.

The interpreter waits, or does not wait, for the reply as specified by the programmer in the *connectMonthCalendarEvent* method. The operating system ignores the value of the reply.

Arguments:

The event handling method receives four arguments:

selStart

A **`DateTime`** object that is the new selected date, or the first selected date in a range of selected dates.

selEnd

A **`DateTime`** object that is the last selected date in a range of selected dates. If only a single date is selected, then *selEnd* will be the same date as *selStart*.

id

The resource ID of the month calendar sending the notification.

hwnd

The window *handle* for the month calendar control sending the notification.

Return:

The programmer can return any value because the operating system ignores the returned value. Typically 0 is returned.

4.7.18.5. ViewChange Event Handler

```
::method onViewChange unguarded
  use arg oldView, newView, id, hwnd

  return 0
```

The view change notification is sent when the current view changes. The notification is only sent on Windows Vista or later. A syntax exception is raised if the VIEWCHANGE event is connected when the program is not running on Vista or later.

The interpreter waits, or does not wait, for the reply as specified by the programmer in the *connectMonthCalendarEvent* method. The operating system ignores the value of the reply.

Arguments:

The event handling method receives four arguments:

oldView

The *oldView* argument is a keyword denoting what the previous view was. It will be one of: month, year, decade, or century.

newView

The *newView* argument is a keyword denoting what the view was changed to. It also will be one of: month, year, decade, or century.

id

The resource ID of the month calendar sending the notification.

hwnd

The window *handle* for the month calendar control sending the notification

4.7.19. connectMove

```
>>--connectMove(--+--------------+--+--------------+--)---------><
                  +--methodName--+  +-,-willReply--+
```

The *connectMove* method connects a dialog move event notification with a method in the Rexx dialog. The notification is sent after the position of the dialog has changed.

Arguments:

The arguments are:

methodName [optional]

The name of the method that will be invoked each time the dialog has moved. The name can not be the empty string. When this argument is omitted the name defaults to *onMove*.

willReply [optional]

The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.false`**.

Return value:

0

No error.

1

An (internal) error prevented the message from being connected to a method.

Details:

Syntax errors are raised when incorrect usage is detected.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if any move events happen.

The underlying dialog receives this event notification as a WM_MOVE message.

Example:

```
::class MyDlgClass subclass UserDialog

::method init
  forward class (super) continue

  self~connectMove(onMove)

::method onMove unguarded
  use arg unUsed, posInfo

  -- The dialog position has changed, print out where we are.
  x = .DlgUtil~loWord(posInfo)
  y = .DlgUtil~hiWord(posInfo)
  say 'At coordinate (' x',' y' ) on the screen. (In pixels.)'

  return 0
```

4.7.19.1. Move Event Handler

The event handler for the Move event is invoked when the position of the dialog has changed.

The *willReply* argument in the *connectMove* method determines how the event handler needs to respond to the notification.

```
::method onMove unguarded
  use arg wParam, lParam

  return 0
```

Arguments:

The event handling method receives 2 arguments:

wParam

The numeric value of the WPARAM parameter of the notification *message*. This argument has no meaning and is likely 0. Even if it is not 0, it still has no meaning.

lParam

The numeric value of the LPARAM parameter of the notivication message. This value contains the new position coordinates of the dialog. The low word of the number contains the new X position and the high word contains new y position. The *loWord* and *hiWord* methods can be used to extract these values.

Return:

The MSDN documentation says to return 0 from the event handler.

Example

A complete working example to play with:

```
/* Simple connectMove() example */

  dlg = .MoveDlg~new
  dlg~execute("SHOWTOP", IDI_DLG_OOREXX)

::requires "ooDialog.cls"

::class 'MoveDlg' subclass UserDialog

::method init
  forward class (super) continue

  self~connectMove(onMove, .true)

  self~create(30, 30, 257, 123, "Move Me Dialog", "CENTER")

::method defineDialog

  self~createPushButton(IDOK, 142, 99, 50, 14, "DEFAULT", "Ok")
  self~createPushButton(IDCANCEL, 197, 99, 50, 14, , "Cancel")

::method onMove unguarded
  use arg unUsed, posInfo

  -- Get the new position ...
  x = .DlgUtil~loWord(posInfo)
  y = .DlgUtil~hiWord(posInfo)

  -- If x is greater than 400, the move us back
  -- to an x position of 50
  if x > 400 then do
    p = .Point~new(50, y)
    self~moveTo(p)
    j = SysSleep(.01)
  end

  return 0
```

4.7.20. connectNotifyEvent

```
>>--connectNotifyEvent(--id--,--event--+---------------+--+--------------+--)--><
                                       +-,--methodName-+  +-,-willReply--+
```

The *connectNotifyEvent* method connects one of the generic event notifications from a dialog control to a method in the Rexx dialog object.

Arguments:

The arguments are:

id [required]

The resource ID of the dialog control for which the notification is to be connected. This can be numeric or symbolic.

event [required]
: Exactly one of the following key words, case is not significant, that specifies which event is to be connected:

  CLICK
  : The left mouse button was clicked on the dialog control.

  DBLCLK
  : The left mouse button was double-clicked on the dialog control.

  ENTER
  : The return key was pressed in the dialog item.

  GOTFOCUS
  : The dialog item got the input focus.

  LOSTFOCUS
  : The dialog item lost the input focus.

  OUTOFMEMORY
  : The dialog control is out of memory.

  RCLICK
  : The right mouse button was clicked on the dialog item.

  RDBLCLK
  : The right mouse button was double-clicked on the dialog control.

methodName [optional]
: The name of the event handling method. This method is invoked each time the specified event occurs. The method name can not be the empty string. If you omit this argument, the event handler method name is generated for you. This name will be the event keyword, preceded by **on**. For example: *onLostFocus*.

willReply [optional]
: The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.false`**.

Return value:

The return codes are:

0
: No error detected.

-1
: The resource ID could not be resolved or the event argument is incorrect.

1
: The message was not connected correctly.

Remarks:

Use of the *connectNotifyEvent method is discouraged*

Most event notifications are specific to the particular type of control that sends them. These generic event notifications are common to a number of dialog controls. In general, if one of the dialog control specific event connection methods, such as *connectListViewEvent*, can make a connection for the event, then *connectNotifyEvent* should not be used. In these cases, the dialog control does not send one of the generic event notifications. It sends a notification specific to itself.

In addition, most of the dialog control specific event connections provide better information to the event handler than that provided by the event handler used for a generic connection.

In Windows itself, these events are sent to the parent dialog using the WM_NOTIFY message.

Details:

Syntax errors are raised when incorrect usage is detected.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if the connected event happen.

In Windows itself, these events are sent to the parent dialog using the WM_NOTIFY message.

Example:

The following example connects the double-click of the left mouse button on dialog control DLGITEM1 with method onDblClk:

```
::class MyDlgClass subclass UserDialog

::method init
  self~init:super(...)
  self~connectNotifyEvent(DLGITEM1, "DBLCLK")

::method onDblClk unguarded
  use arg id, hwnd
  say "Control" id " has been double-clicked! Its window handle is:" hwnd

  return 0
```

4.7.20.1. General Notify Event Handler

All event handlers connected through the *connectNotifyEvent* method are sent the same arguments and are essentially coded the same way.

The *willReply* argument in the *connectNotifyEvent* method determines how the event handler needs to respond to the notification.

```
::method onNotifyEvent unguarded
  use arg id, ptr, notifyCode, controlObj

  return 0
```

Arguments:

The event handling method receives 4 arguments:

id

The numeric resource ID of the dialog control sending the notification.

ptr

The numeric value of a pointer to a memory location in the running application. This is of no conceivable use to the ooDialog programmer, but is retained for backwards compatibility.

notifyCode

The numeric notification code of the event that caused the notification to be sent.

controlObj

The Rexx dialog control object that represents the underlying dialog control that sent the notification. It is possible this will be the **`.nil`** object if some error happened, but this is very unlikely.

Return:

In general, 0 should be returned. For most of the events connected through the *connectNotifyEvent* method, the operating system ignores the returned value.

Nevertheless, beginning in ooDialog 4.2.4, ***if*** the *willReply* argument in the *connectNotifyEvent* method is set to true by the programmer, the ooDialog framework will take the value returned from the event handler and return that value to the operating system. If, through research, the ooDialog programmer determines a better return value for his particular use of *connectNotifyEvent*, then he is encouraged to set *willReply* to true and return the better value from his event handler.

4.7.21. connectPosChanged

```
>>--connectPosChanged(--+--------------+--+--------------+--)---><
                        +--methodName--+  +-,-willReply--+
```

The *connectPosChanged* method connects the position changed event notification sent to a dialog to a method in the Rexx dialog object. This notification is sent to the dialog when its size, position, or place in the Z order has changed.

Arguments:

The arguments are:

methodName [optional]

The name of the method that will be invoked each time the dialog has moved. The name can not be the empty string. When this argument is omitted the name defaults to *onPosChanged*.

willReply [optional]

The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.false`**.

Return value:

0

No error.

1

An (internal) error prevented the message from being connected to a method.

Details:

Syntax errors are raised when incorrect usage is detected.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if any position changed events happen.

In Windows itself, the dialog receives this event notification as a WM_WINDOWPOSCHANGED message.

Example:

```
::class 'MyDlgClass' subclass UserDialog

::method initDialog
  ...
  self~connectPosChanged("onNewPos")

::method onNewPos unguarded
  use arg wp, lp
  rect = self~windowRect
  say "The new dialog window rectangle is:"
  say "  Left:   " rect~left
  say "  Top:    " rect~top
  say "  Right:  " rect~right
  say "  Bottom:" rect~bottom

  return 0
```

4.7.21.1. PosChanged Event Handler

The event handler for the Position Changed event is invoked when the size, position, or place in the Z order of the dialog has changed.

The *willReply* argument in the *connectMove* method determines how the event handler needs to respond to the notification.

```
::method onPosChanged unguarded
  use arg wParam, lParam

  return 0
```

Arguments:

The event handling method receives 2 arguments:

wParam

The numeric value of the WPARAM parameter of the notification *message*. This argument has no meaning and is likely 0. Even if it is not 0, it still has no meaning.

lParam

The numeric value of the LPARAM parameter of the notivication message. This is a pointer to a memory location and has no conceivable use in the ooDialog program.

Return:

The event handler should probably return 0.

4.7.22. connectReBarEvent

```
>>--connectReBarEvent(--id--,--event--+-----------+--+-------------+--)--------><
                                      +-,-mthName-+  +-,-willReply-+
```

Connects an event notification message from a rebar control to a method in the Rexx dialog object.

Arguments:

The arguments are:

id [required]

The resource ID of the rebar control whose notification message is to be connected to a method in the Rexx dialog. May be numeric or *symbolic*.

event [required]

Exactly one of the following keywords. The keyword specifies the event to be connected. Case is not significant.

AUTOBREAK	DELETEDBAND	LAYOUTCHANGED
AUTOSIZE	DELETINGBAND	MINMAX
BEGINDRAG	ENDDRAG	NCHITTEST
CHEVRONPUSHED	GETOBJECT	RELEASEDCAPTURE
CHILDSIZE	HEIGHTCHANGE	SPLITTERDRAG

AUTOBREAK

Notifies the dialog that a break will appear in the bar. The dialog determines whether to make the break.

NCHITTEST

Sent by a rebar control when the control receives a hit test request.

RELEASEDCAPTURE

Notifies the dialog window that the rebar control is releasing the mouse capture.

methodName [optional]

The name of the method that is to be invoked whenever the specified event happens. The programmer defines this method. If this argument is omitted, a method name is automatically generated that consists of the event keyword preceded by **on**. For instance, **onAutoBreak**. The method name can not be the empty string. The empty string is treated as an omitted argument.

willReply [optional]

The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **.true**.

For those event notifications that the operating system expects a return value, the *willReply* argument is ignored. For these events, the event handler must always return a value of the correct type. I.e., if the operating system expects a reply of true or false, the event handler must always return true or false. The events in the following list fall into this category:

NCHITTEST	BEGINDRAG	GETOBJECT
AUTOBREAK	CHILDSIZE	MINMAX

Return value:

Returns **.true** if the event was connected correctly, otherwise **.false**.

Remarks:

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

Details:

Syntax errors are raised when incorrect usage is detected, including the use of an invalid symbolic ID or an unrecognized event keyword.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if the connected event happens.

The underlying dialog receives the RBN_* messages as the notifications for the rebar events.

Example:

This example ...

```

```

4.7.22.1. AutoBreak Event Handler

The event handler for the auto break event is invoked when a break will appear in the bar. The application can allow, or disallow, the break by responding to the notification.

If this event notification is connected, the event handler must respond and return a value to the notification.

```
::method onAutoBreak unguarded
  use arg id, bandIndex, wID, style, reBar, itemData, msgID

  return .true
```

Arguments:

The event handling method receives 7 arguments:

id
: The numeric resource ID of the control that sent the notification.

bandIndex
: The one based index of the band affected by the notification. This is 0 if no band is affected.

wID
: The application-defined ID of the band.

style
: A list of style keywords that specify the current style of the band.

rebar
: The Rexx rebar object that represents the underlying rebar control.

itemData
: The item data assigned to the band, or the **`.nil`** object if there is no item data assigned.

msgID
: TBD what is this?

Return:

The event handler must return true or false. True allows the break to occur and false prevents the break.

Remarks:

xxx

Example

The following example connects several rebar events in the *initDialog* method:

```
::method initDialog

  self~connectReBarEvent(IDC_RBAR, "RELEASEDCAPTURE", onRelease, sync)
  self~connectReBarEvent(IDC_RBAR, "NCHITTEST", onNcHitTest)
  self~connectReBarEvent(IDC_RBAR, "AUTOBREAK", onBreak)

  rb = self~newReBar(IDC_RBAR)
  ...
```

4.7.22.2. NcHitTest Event Handler

The event handler for the non-client hit test event is invoked when the rebar control receives a hit test request. The control then sends the NcHitTest notification to the dialog. The event handler can choose to allow the rebar to perform default processing of the hit test message, or to return its own value to over-ride the default hit test processing.

The event handler must always return a value for this notification.

```
::method onNcHitTest unguarded
  use arg id, bandIndex, pos, hit, rebar

  return 'CAPTION'
```

Arguments:

The event handling method receives 5 arguments:

id
: The numeric resource ID of the control that sent the notification.

bandIndex
: The one based index of the band affected by the notification. This is 0 if no band is affected.

pos
: A *Point* object specifying the mouse coordinates of the hit test message.

hit
: One of the keywords in the *hit test table*.

rebar
: The Rexx rebar object that represents the underlying rebar control that sent the notification.

Return:

Return zero to allow the rebar to perform default processing of the hit test message, or return one of the *hit keywords* to override the default hit test processing.

Remarks:

The NCHITTEST event connection is provided for completeness. It is not entirely clear to the author what the use of the notification is, or what effect over-riding the default processing has.

Example

The following example prints out the values sent by the rebar control for the notification and if the *hit* argument is SIZE returns a non-zero value:

```
::method onNcHitTest unguarded
  use arg id, bandIndex, pos, ht, reBar
  say 'onNcHitTest() id:' id 'bandIndex:' bandIndex 'pos:' pos 'ht:' ht 'reBar:' reBar

  if ht == 'SIZE' then return 'NOWHERE'
  else return 0

/* Output might be for example:

onNcHitTest() id: 1003 bandIndex: 5 pos: a Point (233, 13) ht: CAPTION reBar: a ReBar
onNcHitTest() id: 1003 bandIndex: 5 pos: a Point (230, 14) ht: SIZE reBar: a ReBar
onNcHitTest() id: 1003 bandIndex: 5 pos: a Point (227, 14) ht: SIZE reBar: a ReBar
onNcHitTest() id: 1003 bandIndex: 5 pos: a Point (225, 15) ht: SIZE reBar: a ReBar
onNcHitTest() id: 1003 bandIndex: 5 pos: a Point (223, 15) ht: SIZE reBar: a ReBar
onNcHitTest() id: 1003 bandIndex: 0 pos: a Point (222, 16) ht: CLIENT reBar: a ReBar
onNcHitTest() id: 1003 bandIndex: 4 pos: a Point (220, 16) ht: CAPTION reBar: a ReBar
onNcHitTest() id: 1003 bandIndex: 4 pos: a Point (219, 16) ht: CAPTION reBar: a ReBar
*/
```

4.7.22.3. ReleasedCapture Event Handler

The event handler for the released capture event is invoked the rebar control has released the mouse capture.

The *willReply* argument in the *connectReBarEvent* method determines how the event handler needs to respond to the notification.

```
::method onReleasedCapture unguarded
  use arg id, rebar

  return 0
```

Arguments:

The event handling method receives 2 arguments:

id

The numeric resource ID of the control that sent the notification.

rebar

The Rexx rebar object that represents the underlying rebar control that sent the notification.

Return:

The operating system ignores the return from this notification. 0 makes a good value to return.

Example

The following example prints out the values of the arguments sent by the rebar control for the released capture event:

```
::method onRelease unguarded
  use arg id, reBar
  say 'onRelease() id:' id 'reBar:' reBar
  return 0

/* Output might be:

onRelease() id: 1003 reBar: a ReBar
onRelease() id: 1003 reBar: a ReBar
onRelease() id: 1003 reBar: a ReBar
*/
```

4.7.23. connectResize

```
>>--connectResize(--+--------------+--+--------------+--)-------><
                    +--methodName--+  +-,-willReply--+
```

The *connectResize* method connects a size event notification to the underlying dialog with a method in the Rexx dialog. The notification is sent to the dialog after its size has changed.

Arguments:

The arguments are:

methodName [optional]

The name of the method that is invoked each time the size of the dialog has changed. The method name can not be the empty string. When this argument is omitted the name defaults to *onResize*.

willReply [optional]

The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`SYNC`**. See the *Remarks* section for some discussion on why the default is not false as it normally would be for older event connection methods.

synch [optional]

A boolean value that specifies if the interpreter should invoke the event handler *directly*, or not. The default is **`.true`**, see the remarks. If this behavior is not desired, the *synch* argument can be used to specify that the interpreter should not wait for a reply.

Return value:

0

No error.

1

An (internal) error prevented the message from being connected to a method.

Remarks:

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

Typically when the user is resizing a window, there are many event notifications that come quickly. In older ooDialog versions, where the notification was placed on a queue to be processed at a later date, this caused a number of notifications to be placed on the queue before a single notification was processed. Programs that connected the resize event performed poorly when many notifications pile up before they are processed.

The poor performance is mostly fixed by having the interpreter invoke the event handler directly and wait for the return from the event handler, which causes each notification to be processed one by one. When *willReply* is set to *SYNC*, all ooDialog programs that connect the resize event are likely to perform much better. This is why the default for *willReply* is set to *SYNC*

Normally, when the interpreter waits for the event handler to return, a syntax condition is raised if a value is not returned from the event handler. The *SYNCH* option causes the interpreter to wait until the event handler returns, but to not expect a returned value. This provides backwards compatibility for older programs where the programmer ***mistakenly*** did not return a value from event handlers. Best practice is to *always* return a value from an event handler. If the operating system ignores the return value, return 0.

Note that when the *ResizingAdmin* is inherited by a dialog to produce a resizable dialog, the *connectResize* method has no effect. The event notification events are handled internally by the **`ResizingAdmin`** object.

Details:

Syntax errors are raised when incorrect usage is detected.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if any size events happen.

The underlying dialog receives the WM_SIZE message as the notification for this event.

Example:

```
  dlg = .ResizingDialog~new
  dlg~createCenter(100, 60, "Resize Me", "THICKFRAME")
  dlg~execute("SHOWTOP")

::requires 'ooDialog.cls'

::class 'ResizingDialog' subclass UserDialog

::method init
  forward class (super) continue

  self~connectResize("onSize")

::method onSize unguarded
  use arg sizeEvent, sizeInfo

  -- sizeInfo contains information about the new width and height in pixels.
  w = .DlgUtil~loWord(sizeinfo)
  h = .DlgUtil~hiWord(sizeinfo)
  say "New width=" w ", new height=" h

  return 0
```

4.7.23.1. Resize Event Handler

The event handler for the Resize event is invoked when the size of the dialog has changed.

The *willReply* argument in the *connectResize* method determines how the event handler needs to respond to the notification.

```
::method onResize unguarded
  use arg wParam, lParam

  return 0
```

Arguments:

The event handling method receives 2 arguments:

wParam

A numeric code that specifies the type of resizing. The dialog object provides a number of *constant* values to make it easier to decode the numeric value of the *wParam* argument. The constants all begin with *SIZE_*.

lParam

The numeric value of the LPARAM parameter of the notivication message. This value contains the new size coordinates of the dialog. The low word of the number contains the new width and the high word contains new height. The *loWord* and *hiWord* methods can be used to extract these values.

Return:

The MSDN documentation says to return 0 from the event handler.

Example:

This example is pulled from the File Viewer *example* at the end of the *Appearance and Behavior Methods* section. A complete working example is presented there that uses a number of the dialog methods:

```
::method defineDialog
  expose wasMinimized

  wasMinimized = .false
  style = "VSCROLL HSCROLL MULTILINE READONLY"
  self~createEdit(IDC_MULTILINE, 0, 0, 170, 180, style, "cEntry")
  self~connectResize("onSize", .true)
  ...

/* The first arg, sizeEvent, is a flag that the OS sends specifying the type of
 * size event.  We are only interested in these 3 flags:
 *
 * SIZE_RESTORED   = 0
 * SIZE_MINIMIZED  = 1
 * SIZE_MAXIMIZED  = 2
 */
::method onSize unguarded
  expose wasMinimized
  use arg sizeEvent, sizeInfo

  if sizeEvent == self~SIZE_MINIMIZED then wasMinimized = .true

  if sizeEvent == self~SIZE_RESTORED |  sizeEvent == self~SIZE_MAXIMIZED then do
    if \ wasMinimized then self~resizeEditControl
    wasMinimized = .false
  end

  return 0
```

4.7.24. connectResizing

```
>>--connectResizing(--+--------------+--+-------------+--)-----><
                      +--methodName--+  +-,-willReply--+
```

Connects a *sizing event* notification sent to the underlying dialog with a method in the Rexx dialog. The notification is sent to the dialog *before* its size has changed.

Arguments:

The arguments are:

methodName [optional]

The name of the method that is invoked each time the size of the dialog is about to be changed. The method name can not be the empty string. When this argument is omitted the name defaults to *onResizing*.

willReply [optional]

Although the *willReply* argument is accepted so that the *connectResizing* method is similar to the other event connection methods, its value is always considered to be true. That is, the interpreter waits for the return value from the event handler. The event handler must always return true or false.

Return value:

0

No error.

1

An (internal) error prevented the message from being connected to a method.

Remarks:

By processing the sizing event notification, the programmer can monitor the size and position of the drag rectangle of the dialog being resized and, if desired, can change its new size or position.

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

The interpreter invokes the event handler directly and waits in the window *message* processing loop for the return from the event handler. Connecting the sizing event requires that the programmer reply to the event from the event handler in a timely manner.

Note that when the *ResizingAdmin* is inherited by a dialog to produce a resizable dialog, the *connectResizing* method has no effect. The event notification events are handled internally by the **`ResizingAdmin`** object.

Details:

Syntax errors are raised when incorrect usage is detected.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if any sizing events happen.

The underlying dialog receives the WM_SIZING message as the notification for this event.

Example:

This example shows, partially, how to prevent a resizable dialog from being sized smaller than 300 pixels high. A complete implementation would also have to handle the TOPLEFT, TOPRIGHT, etc., directions. And, normally, an implementation would also enforce a minimum width.

A complete implementation can be found in the **`dlgAreaUDemoThree.rex`** example program.

```
::method init
  ...

  self~connectResizing("onSizing", .true)

::method onSizing unguarded
  use arg rect, direction
```

```
select
  when direction == 'TOP' then do
    if rect~bottom - rect~top < 300 then do
      rect~top = rect~bottom - 300
      return .true
    end
  end
  when direction == 'BOTTOM' then do
    if rect~bottom - rect~top < 300 then do
      rect~bottom = rect~top + 300
      return .true
    end
  end
  ...
  otherwise
    nop
end

return .false
```

4.7.24.1. Resizing Event Handler

The event handler for the Resizing event is invoked when the size of the dialog is about to be changed. That is, it is invoked with arguments specifying the new size of the dialog, but before the change is actually made.

The operating system expects the application to reply to this message, either to allow the change or to disallow the change. The Rexx event hander must reply to this notification, this can not be changed.

```
::method onResize unguarded
  use arg wParam, lParam

  return .false
```

Arguments:

The event handling method receives 2 arguments:

rect

A **`Rect`** object that specifies the size the dialog is about to be changed to.

direction

A keyword that specifies which edge of the dialog window is being sized. The keyword will be exactly one of the following:

BOTTOM	LEFT	TOPLEFT
BOTTOMLEFT	RIGHT	TOPRIGHT
BOTTOMLEFT	TOP	

BOTTOM

The bottom edge is being dragged.

BOTTOMLEFT

The bottom-left corner is being dragged.

BOTTOMRIGHT

The bottom-right corner is being dragged.

LEFT
: The left edge is being dragged.

RIGHT
: The right edge is being dragged.

TOP
: The top edge is being dragged.

TOPLEFT
: The top-left corner is being dragged.

TOPRIGHT
: The top-right corner is being dragged.

Return:

Return false to indicate the coordinates specified in the *rect* arg are unchanged. Otherwise, return true to indicate the coordinates have been changed and the operating system will set the new size of the dialog to the changed coordinates.

Remarks:

The application can effect how the dialog is resized by changing the values of the *rect* argument and returning true. The operating updates the size of the drag rectangle with the new values in the *rect* argument rather than with the original values. Typically this is done to prevent the user from resizing the dialog too small or too big.

Example:

This example shows an event handler that prevents the dialog from being sized smaller than a minimum size, but puts no restriction on the maximum size.

```
::method onResizing unguarded
  expose minWidth minHeight
  use arg r, direction

  select
    when direction == 'TOP' then do
      if minHeight > r~bottom - r~top then do
        r~top = r~bottom - minHeight
        return .true
      end
    end
    when direction == 'BOTTOM' then do
      if minHeight > r~bottom - r~top then do
        r~bottom = r~top + minHeight
        return .true
      end
    end
    when direction == 'LEFT' then do
      if minWidth > r~right - r~left then do
        r~left = r~right - minWidth
        return .true
      end
    end
    when direction == 'RIGHT' then do
      if minWidth > r~right - r~left then do
        r~right = r~left + minWidth
        return .true
      end
    end
    when direction == 'BOTTOMLEFT' then do
      didChange = .false
      if minHeight > r~bottom - r~top then do
        r~bottom = r~top + minHeight
        didChange = .true
      end

      if minWidth > r~right - r~left then do
        r~left = r~right - minWidth
        didChange = .true
      end

      return didChange
    end
    when direction == 'BOTTOMRIGHT' then do
      didChange = .false

      if minHeight > r~bottom - r~top then do
        r~bottom = r~top + minHeight
        didChange = .true
      end

      if minWidth > r~right - r~left then do
        r~right = r~left + minWidth
        didChange = .true
      end

      return didChange
    end
    when direction == 'TOPLEFT' then do
      didChange = .false

      if minHeight > r~bottom - r~top then do
        r~top = r~bottom - minHeight
        didChange = .true
      end

      if minWidth > r~right - r~left then do
        r~left = r~right - minWidth
        didChange = .true
      end

      return didChange
    end
    when direction == 'TOPRIGHT' then do
      didChange = .false

      if minHeight > r~bottom - r~top then do
        r~top = r~bottom - minHeight
        didChange = .true
      end

      if minWidth > r~right - r~left then do
        r~right = r~left + minWidth
        didChange = .true
      end

      return didChange
    end
    otherwise
      nop
  end
  -- End select
  return .false
```

4.7.25. Connecting Scroll Bar Events

For connecting scroll bars, ooDialog provides the typical *connectScrollBarEvent* method along with 2 convenience methods.

4.7.25.1. connectAllSBEvents

```
>>--connectAllSBEvents(-id-,-mName-+------+-+------+-+-----+-+--------------+-)--><
                                   +-,-mn-+ +-,-mx-+ +-,-p-+ +-,-willReply--+
```

Connects all scroll bar events to one method.

Arguments:

The arguments are:

id [required]
: The resource ID of the scroll bar.

mName [required]
: The method that is invoked for all events sent by the scroll bar. There is no default name for the method, the name must be specified.

mn, mx [optional]
: Sets the minimum and maximum values for the range of the scroll bar.

p [optional]
: Sets the current position of the scroll bar.

willReply [optional]
: The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.false`**.

Return value:

-1
: The specified symbolic ID could not be resolved.

0
: No error.

Remarks:

This method requires that the *underlying* scroll bar exists. Therefore, the method is best invoked in the *initDialog* method.

See the sections on *connecting* and *coding* event handlers for additional information on event handlers. The section for the *SCROLL* event handler contains information specific to the event handler for scroll bar events.

4.7.25.2. connectEachSBEvent

```
>>--connectEachSBEvent(-id-,-mthWhenUp-,-mthWhenDown-+---------------+--------->
                                                     +-,-mthWhenDrag-+
>--+-------+-+-------+-+-------+--+-----------+-+-----------+-+----------+----->
   +-,-min-+-+-,-max-+ +-,-pos-+  +-,-mthPgUp-+ +-,-mthPgDn-+ +-,-mthTop-+

>--+-------------+-+------------+-+------------+-+-------------+--)------------><
   +-,-mthButtom-+ +-,-mthTrack-+ +-,-mthEndSc-+ +-,-willReply-+
```

The *connectEachSBEvent* method connects an *event* notification message from a scroll bar to a method in the Rexx dialog object. Optionally, the method can also initialize the scroll bar, (set the range and current position.)

Arguments:

The arguments are:

id [required]
: The ID of the scroll bar.

mthWhenUp [required]
: The method that is called each time the scroll bar is incremented.

mthWhenDown [required]
: The method that is called each time the scroll bar is decremented.

mthWhenDrag [optional]
: The method that is called each time the scroll bar is dragged with the mouse.

min, max [optional]
: The minimum and maximum values for the scroll bar.

pos [optional]
: The current or preselected value.

mthPgUp [optional]
: The method that is called each time the scroll bar is focused and the PgUp key is pressed.

mthPgDn [optional]
: The method that is called each time the scroll bar is focused and the PgDn key is pressed.

mthTop [optional]
: The method that is called each time the scroll bar is focused and the Home key is pressed.

mthBottom [optional]
: The method that is called each time the scroll bar is focused and the End key is pressed.

mthTrack [optional]
: The method that is called each time the scroll box is dragged.

mthEndSc [optional]
: The method that is called each time the scroll box is released after the dragging.

willReply [optional]
: The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.false`**.

Return value:

-1
: The specified symbolic ID could not be resolved.

0
: No error.

Remarks:

This method requires that the *underlying* scroll bar exists. Therefore, the method is best invoked in the *initDialog* method.

See the sections on *connecting* and *coding* event handlers for additional information on event handlers. The section for the *SCROLL* event handler contains information specific to the event handler for scroll bar events.

Example:

In the following example, scroll bar with symbolic ID IDC_SB is connected to three methods. The range is initialized to 1 as the minimum and 20 as the maximum. The current position is set to 6:

```
::class MyDialog subclass UserDialog
  ...
::method initDialog

  self~connectEachSBEvent(IDC_SB,"Increase", "Decrease", "Drag", 1, 20, 6)
```

4.7.25.3. connectScrollBarEvent

```
>>--connectScrollBarEvent(--id--,--event--+---------+-+-------------+--)-------><
                                          +-,-mName-+ +-,-willReply-+
```

Connects a method in the Rexx dialog to the Windows *event* notification from a scroll *ScrollBar* control.

Arguments:

The arguments are:

id [required]

The resource ID of the scroll bar whose event notification is to be connected. May be numeric or *symbolic*.

event [required]

A keyword specifying which scroll event should be connected. This methods connects either a vertical or a horizontal scroll bar. The symbolic keyword names are orientated towards which type the scroll bar is. But the values that the scroll bar sends are the same for similar events.

I.e., one event is to move 1 unit towards the upper left. For a vertical scroll bar this is one line up (LINEUP), for a horizontal scroll bar this is one unit to the left (LINELEFT). But, the numeric value that a scroll bar sends for the event is the same. Therefore, the LINEUP and LINELEFT keywords produce the identical effect. If the scroll bar is a vertical scroll bar, using the LINELEFT keyword produces the same effect as using the LINEUP keyword. The programmer can use whichever keyword makes the most sense.

The keyword must be one of the following, case is not significant:

LINEUP	PAGELEFT	TOP
LINELEFT	PAGEDOWN	LEFT
LINEDOWN	PAGERIGHT	BOTTOM
LINERIGHT	POSITION	RIGHT

PAGEUP	DRAG	ENDSCROLL

LINEUP LINELEFT
: The scroll bar was scrolled to the left or up by one unit.

LINEDOWN LINERIGHT
: The scroll bar was scrolled to the right or down by one unit.

PAGELEFT PAGEUP
: The scroll bar was scrolled to the left or up by one page size.

PAGERIGHT PAGEDOWN
: The scroll bar was scrolled to the right or down by one page size.

DRAG
: The user is dragging the scroll box. This message is sent repeatedly until the user releases the mouse button.

TOP LEFT
: The scroll bar was scrolled completely to the top or to the left.

BOTTOM RIGHT
: The scroll bar was scrolled completely to the bottom or to the right.

ENDSCROLL
: Scrolling has been ended, that is, the appropriate key or mouse button has been released.

POSITION
: The scroll box (thumb) of the scroll bar was dragged and the user has released the mouse button.

mName [optional]
: The name of the method that is to be invoked whenever the specified notification is received from the scroll bar. If this argument is omitted, the method name is automatically generated by concatenating *on* with the event keyword. For example, *onLineUp*.

willReply [optional]
: The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.false`**.

Return value:

The return codes are:

0
: No error detected.

-1
: The resource ID could not be resolved or the event argument is incorrect.

1
: The messages was not connected correctly.

Remarks

This method requires that the *underlying* scroll bar exists. Therefore, the method is best invoked in the *initDialog* method. At some later point in the life cycle of the dialog is fine. If the method is invoked before the underlying dialog has been created a syntax condition is raised.

See the sections on *connecting* and *coding* event handlers for additional information on event handlers. The section for the *SCROLL* event handler contains information specific to the event handler for scroll bar events.

Example:

The following example connects the POSITION scroll event with a method in the Rexx dialog. Since the third argument is omitted, the ooDialog framework automatically connects the event to the *onPosition* method. The event handler extracts the new position from the notification arguments and sets the position for the scroll bar. The easist way to do this is to use the *determinePosition* method.

This example also shows how to calculate the new position without using the *determinePosition* method and then displays the new position and the event type. The **ScrollBar** class provides *Scroll Event Constants*, one of which is the THUMBPOSITION constant, whose value is the value of the POSITION event code.

```
::class 'SimpleDialog' subclass UserDialog

::method initDialog

  self~connectScrollBarEvent(IDC_SB_FILE, "POSITION")

::method onPosition unguarded
  use arg posInfo, hwnd, scrollBar

  -- Set the scroll bar to the new position and have it redraw itself.
  newPos = scrollBar~determinePosition(posInfo)
  scrollBar~setPos(newPos, .true)

  -- Determine the new position ourself and the display it for comparison
  pos       = .DlgUtil~hiWord(posInfo)
  eventCode = .DlgUtil~loWord(posInfo)

  say "New Position:     " pos
  say "Verify event code:"
  say "  THUMBPOSITION:" .ScrollBar~THUMBPOSITION
  say "  This event:    " eventCode

  return 0
```

4.7.25.4. Scroll Event Handler

The event handler for the Scroll event is invoked when the user moves the scroll box (the thumb box) in a scroll bar.

The *willReply* argument in the method connecting the scroll event, *connectScrollBarEvent*, *connectAllSBEvents*, etc., determines how the event handler needs to respond to the notification.

```
::method onScroll unguarded
  use arg posInfo, hwnd, scrollBar

  return 0
```

Arguments:

The event handling method receives 3 arguments:

posInfo

The *posInfo* argument contains the scroll event code in the low word of the argument. For the POSITION and the DRAG scroll events only, the high word contains the position of the scroll box. For the other events, the high word is not used.

hwnd

When the notification comes from a *ScrollBar* bar control, the *hwnd* argument is the window handle of the scroll bar. When the notification does not come from a scroll bar control, *hwnd* is 0.

scrollBar

When the notification comes from a *ScrollBar* bar control, the *scrollBar* argument is the Rexx **`ScrollBar`** object representing the underlying scroll bar. When the notification does not come from a scroll bar control, *scrollBar* will be the **`.nil`** object.

Return:

The operating system ignores the return from this notification so 0 makes a good return value.

Remarks:

When the user interacts with a scroll bar, the operating system does not reposition the scroll box (the thumb box.) Rather, the operating system relies on the application, (in essence the programmer,) to update the scroll bar position during the scroll bar event notification. Therefore, if the scroll bar event is not connected, or the programmer does not reposition the scroll box in the event handler, it will appear to the user that the scroll box can not be moved.

The *ScrollBar* class provides the *determinePosition* method as a convenience to determine the new position of the scroll bar. To use this method, pass the *posInfo* argument to the method:

```
::method onScroll unguarded
  use arg posInfo, hwnd, scrollBar

  -- Set the scroll bar to the new position and have it redraw itself.
  newPos = scrollBar~determinePosition(posInfo)
  scrollBar~setPos(newPos, .true)
```

The **`ScrollBar`** class provides *Scroll Event Constants*, which can be helpful in determining the new position without using the *determinePosition* method

Example

The following example connects all the scroll events to a single event handler. In the event handler the *determinePosition* method is used to find the new positiona and the scroll bar is updated with the new postion:

```
::class 'FileReader' subclass UserDialog

...

::method initDialog

  self~connectAllSBEvents(IDC_SB_FILE, onScroll)

::method onScroll unguarded
  use arg posInfo, hwnd, scrollBar

  -- Set the scroll bar to the new position and have it redraw itself.
  newPos = scrollBar~determinePosition(posInfo)
  scrollBar~setPos(newPos, .true)
```

```
    return 0
```

4.7.26. connectSizeMoveEnded

```
>>--connectSizeMoveEnded(--+-------------+--+-------------+--)---------------><
                           +--methodName--+  +-,-willReply--+
```

The *connectSizeMoveEnded* method connects the Windows exit size / move *event* with an event handling method in the Rexx dialog object. This event is sent exactly one time when the user has stopped moving or stopped resizing the dialog.

Arguments:

The arguments are:

methodName [optional]

The name of the event handling method that to be invoked when the size / move exit event occurs. The name must not be the empty string. When this argument is omitted the name defaults to *onSizeMoveEnded*.

willReply [optional]

The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.true`**.

Return value:

Returns 0 on success and 1 if an (internal) error prevented the message from being connected to a method.

Remarks:

Unlike most event handlers, the event handling method for the size / move ended event does not receive any arguments. The Windows operating system ignores the return value to the event notification.

Note that when the *ResizingAdmin* is inherited by a dialog to produce a resizable dialog, the *connectSizeMoveEnded* method has no effect. The event notification events are handled internally by the **`ResizingAdmin`** object.

Details:

Syntax errors are raised when incorrect usage is detected.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if any exit size / move events happen.

In Windows itself, the dialog receives this notification as a WM_EXITSIZEMOVE message.

Example:

This example comes from a resizable dialog implemented through the *DlgAreaU* class. It turns off forcing the dialog controls to redraw in the **`DlgAreaU`** object, and only has the controls redraw once when the sizing is finished.

```
::method init
    expose sizing

    self~init:super
    ...
    self~connectResize('onResize')
    self~connectSizeMoveEnded('onExitSizeMove')
```

```
     sizing = .false
     ...


  ::method defineDialog
    expose u

    u = .dlgAreaU~new(self)
    u~updateOnResize = .false


  ::method onResize unguarded
    expose u sizing
    use arg ignored, sizeinfo

    sizing = .true

    u~resize(self, sizeinfo)
    return 0

  ::method onExitSizeMove unguarded
      expose sizing

      if sizing then do
        -- The user has stopped sizing the dialog, we will now show all the dialog
        -- controls we hid previously, and force the controls to redraw.
        self~showAllControls
        self~update
        sizing = .false
      end

      return 0
```

4.7.26.1. SizeMoveEnded Event Handler

The event handler for the SizeMoveEnded event is invoked once, and only once, when the user has ended moving or ended sizing a window.

The *willReply* argument in the *connectMove* method determines how the event handler needs to respond to the notification.

```
::method onSizeMoveEnded unguarded
  use arg

  return 0
```

Arguments:

The event handling method does not receive any arguments.

Return:

The operating system ignores any return from this notification. 0 make a good return value.

Example:

This example uses the resize and the size move ended event together. Rather than redraw the dialog controls for every new size while the user is dragging the dialog border, it waits and only redraws the controls when the user finishes resizing. To do this it sets a flag when the user starts to resize the dialog and then waits for the size move ended event to know that the user has finished:

```
::method onResize unguarded
```

```
  expose u sizing minMaximized lastSizeInfo
  use arg sizingType, sizeinfo

  -- Save the size information so we know the final size of the dialog.
  lastSizeInfo = sizeInfo

  -- The size / move ended event does not occur when the user maximizes,
  -- minimizes, or restores from maximized / minimized.  Because of that, we
  -- need to redraw the client area under those conditions.

  if sizingType == self~SIZE_MAXIMIZED | sizingType == self~SIZE_MINIMIZED then do
    minMaximized = .true
    if sizingType == self~SIZE_MAXIMIZED then do
      u~resize(self, sizeinfo)
      self~redrawClient(.true)
    end
  end
  else if sizingType == self~SIZE_RESTORED, minMaximized then do
    minMaximized = .false
    u~resize(self, sizeinfo)
    self~redrawClient(.true)
  end
  else do
    -- We are resizing now.
    sizing = .true
  end

  return 0


::method onSizeMoveEnded unguarded
  expose u sizing lastSizeInfo

  -- If we were resizing, force the dialog controls to redraw themselves.
  if sizing then do
    u~resize(self, lastSizeInfo)
    self~redrawClient(.true)
  end

  -- We are not resizing anymore.
  sizing = .false
  return 0
```

4.7.27. connectStaticEvent

```
>>--connectStaticNotify(--id--,--event--,-+----------+--+-------------+--)----><
                                          +-,--mName-+  +-,-willReply--+
```

connectStaticEvent connects a notification message from a static control to a method, defined by the programmer, in the Rexx dialog object. Normally, static controls do not send notification messages. A static control will only send the messages when it has the NOTIFY style. The notification messages inform the dialog that an event has occurred with regard to the static control.

For user defined dialogs use the NOTIFY style keyword in the *create...* static control methods when the control is defined. For dialogs created from a compiled resource or a resource script file use the SS_NOTIFY style when defining the control in a resource editor.

Arguments:

The arguments are:

id [required]

The resource ID of the static control. May be numeric or symbolic.

event [required]

A keyword specifying the event to be connected with a method:

CLICK

The static control has been clicked with the mouse.

DBLCLK

The static control has been double-clicked with the mouse.

DISABLE

The static control has been disabled.

ENABLE

The static control has been enabled.

mName [optional]

The method that is to be invoked whenever the specified notification is received from the static control. The programmer defines this method. The method name can not be the empty string. If this argument is omitted, a method name is automatically generated that consists of the event keyword preceded by **on**. For instance, **onClick**.

willReply [optional]

The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **.false**.

Return value:

The return codes are:

0

No error detected.

-1

The resource ID could not be resolved or the event argument is incorrect.

1

The message was not connected correctly. The message was not connected

Remarks:

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

Example:

The following example comes from an application that displays employee statistics. A single click on the employee number field advances the display to the next employee. A double click on either the employee name or employee job duties allows those fields to be edited.

```
::method initDialog

  self~connectStaticEvent(IDC_ST_EMPNO, "CLICK", empLookup)

  self~connectStaticEvent(IDC_ST_EMPJOB, "DBLCLK", editStats)
  self~connectStaticEvent(IDC_ST_EMPNAME, "DBLCLK", editStats)

  first = self~initDatabase
  self~setStats(first)
```

In this example, (a continuation of the above example,) the control ID determines if the user has double clicked on the employee job duties field, or the employee name field. The first 2 and the fourth args are not needed or used so they are just discarded. The last arg, the Rexx static control object is passed on the userUpdate() method:

```
::method editStats unguarded
  use arg , , id, , staticCtrl

  rec = self~getCurrentRecord

  if self~userUpdate(id, staticCtrl, rec) then self~setStats(rec)

  return 0
```

4.7.27.1. General Static Event Handler

The event handler for all the static control events receives the same arguments and is coded in the same fashion. When the handler is invoked is fairly clear from the name of the event. The CLICKED event handler is invoked when the static control is clicked, etc..

The *willReply* argument in the *connectStaticEvent* method determines how the event handler needs to respond to the notification.

```
::method onStaticEvent unguarded
  use arg info, hwnd, id, notifyCode, staticObj

  return 0
```

Arguments:

The event handling method receives 5 arguments. The first and second arguments need to be retained for backwards compatibility, but the last 3 arguments are really what is needed:

info

A numeric value that contains info about the event. The low order word contains the resource ID of the static control sending the event. The high order word contains the static control event code. However, the ooDialog framework now extracts these values for you and sends them as the third and fourth arguments.

hwnd

The window handle of the static control that sent the notification.

id

The numeric resource id of the static control sending the notification.

notifyCode

The numeric notification code of the event that caused the notification to be sent. Each dialog control has its own specific notification codes.

controlObj

The Rexx static control object that represents the underlying dialog control that sent the notification. It is possible this will be the **`.nil`** object if some error happened, but this is very unlikely.

Return:

The operating system ignores the return from this notification. 0 makes a good return value.

Example

The following example comes from an application that uses a static image control. When the user double clicks on the static control, the application switches to the next application mode and the image on the control switches to the icon that represents the current mode of the application:

```
::method defineDialog
  expose someIcon questionIcon isQuestion

  self~createStaticImage(IDC_ICON_QUESTION, 14, 17, 20, 20, "NOTIFY ICON SIZEIMAGE")
  self~createStaticText(IDC_ST_QUESTION, 59, 17, 122, 20)
  self~createPushButton(IDOK, 126, 74, 50, 14, "DEFAULT", 'Ok')

  self~connectStaticEvent(IDC_ICON_QUESTION, "DBLCLK", onDoubleClick, .true)

  questionIcon = .Image~getImage(IDI_QUESTION, ICON)
  ...

::method onDoubleClick unguarded
  use arg info, hwnd, id, notifyCode, static

  self~swapIcons(static)
  return 0
```

4.7.28. connectStatusBarEvent

```
>>--connectStatusBarEvent(--id--,--event--+-----------+--+-------------+--)----><
                                          +-,-mthName-+  +-,-willReply-+
```

Connects an event notification message from a status bar control to a method in the Rexx dialog object.

Arguments:

The arguments are:

id [required]

The resource ID of the status bar control whose notification message is to be connected to a method in the Rexx dialog. May be numeric or *symbolic*.

event [required]

Exactly one of the following keywords. The keyword specifies the event to be connected. Case is not significant.

CLICK	RCLICK	SIMPLEMODECHANGE
DBLCLK	RDBLCLK	

CLICK

Sent by a status bar control to notify the dialog that the user has clicked the left mouse button within the control. The event handler returns true to indicate that the mouse click was handled and to suppress the default processing by the operating system. Return false to allow default processing of the click.

DBLCLK

Sent by a status bar control to notify the dialog that the user has double-clicked the left mouse button within the control. The event handler returns true to indicate that the mouse click was handled and to suppress the default processing by the operating system. Return false to allow default processing of the click.

RCLICK

Sent by a status bar control to notify the dialog that the user has clicked the right mouse button within the control. The event handler returns true to indicate that the mouse click was handled and to suppress the default processing by the operating system. Return false to allow default processing of the click.

RDBLCLK

Sent by a status bar control to notify the dialog that the user has double-clicked the right mouse button within the control. The event handler returns true to indicate that the mouse click was handled and to suppress the default processing by the operating system. Return false to allow default processing of the click.

SIMPLEMODECHANGE

Sent by a status bar control when the simple mode changes due to the invocation of the *simple* method. The reply from the event handler is ignored by the operating system.

methodName [optional]

The name of the method that is to be invoked whenever the specified event happens. The programmer defines this method. If this argument is omitted, a method name is automatically generated that consists of the event keyword preceded by **`on`**. For instance, **`onClick`**. The method name can not be the empty string. The empty string is treated as an omitted argument.

willReply [optional]

The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.true`**.

For those event notifications that the operating system expects a return value, the *willReply* argument is ignored. For these events, the event handler must always return a value of the correct type. I.e., if the operating system expects a reply of true or false, the event handler must always return true or false. The events in the following list fall into this category:

CLICK	RCLICK
DBLCLK	RDBLCLK

Return value:

Returns **`.true`** if the event was connected correctly, otherwise **`.false`**.

Remarks:

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

Details:

Syntax errors are raised when incorrect usage is detected, including the use of an invalid symbolic ID or an unrecognized event keyword.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if the connected event happens.

The underlying dialog receives the SBN_* messages as the notifications for the status bar events.

Example

The following example shows the programmer connecting left and right mouse clicks, along with a simple mode change to the status bar in the application:

```
::method initDialog
```

```
self~connectStatusBarEvent(IDC_STATUSBAR, "RCLICK", onRightClick)
self~connectStatusBarEvent(IDC_STATUSBAR, "CLICK", onClick)
self~connectStatusBarEvent(IDC_STATUSBAR, "SIMPLEMODECHANGE", onModeChange)
```

4.7.28.1. General StatusBar Mouse Click Event Handler

A number of the statusbar events related to mouse clicks are processed internally by ooDialog in the same manner. The event handlers for these events are sent the same arguments and are essentially coded the same way. This *general* event handler is described here for the CLICK, DBLCLK, RCLICK, and RDBLCLK events. The description for the event keyword in the *connectStatusBarEvent* documentation details when each event will invoke this event handler.

The event handler must always reply either true or false to respond to these notification.

```
::method onStatusBarClick unguarded
  use arg id, nCode, index, pt, statusBar

  return .true
```

Arguments:

The event handling method receives 5 arguments:

id

The numeric resource ID of the statusbar sending the notification.

notifyCode

The numeric notification code of the event that caused the notification to be sent. Each dialog control has its own specific notification codes.

index

The one-based index of the section that was clicked. When a status bar is in simple mode there is only one part and the index will always be one.

pt

A *Point* object that specifies the location of the click. Note that MSDN says this point is in screen *coordinates*, but from experimentation it appears to actually be in client coordinates.

statusBar

The Rexx status bar object that represents the underlying dialog control that sent the notification. It is possible this will be the **`.nil`** object if some error happened, but this is very unlikely.

Return:

The event handler must return true or false. Return true to indicate that the mouse click was handled and suppress default processing by the operating system. Return false to allow default processing of the click.

Example

The following example has a status bar divided into 4 sections. The fourth section allows the user to toggle between simple status bar mode and normal by right clicking on section 4. Note that when the status bar is in simple mode, the user can click anywhere to take the status bar out of simple mode:

```
::method onRightClick unguarded
```

```
  use arg id, nCode, index, pt, sb

  if sb~isSimple then do
    sb~simple(.false)
  end
  else if index == 4 then do
    sb~simple(.true)
  end

  return .true
```

4.7.28.2. SimpleModeChange Event Handler

The event handler for the simple mode change event is invoked when the simple mode of the statusbar changes due to a *simple* message.

The *willReply* argument in the *connectStatusBarEvent* method determines how the event handler needs to respond to the notification.

```
::method onModeChange unguarded
  use arg id, code, sb

  return 0
```

Arguments:

The event handling method receives 3 arguments:

id

The numeric resource ID of the statusbar sending the notification.

notifyCode

The numeric notification code of the event that caused the notification to be sent.

controlObj

The Rexx dialog control object that represents the underlying dialog control that sent the notification. It is possible this will be the **`.nil`** object if some error happened, but this is very unlikely.

Return:

The operating system ignores the return from this notification. 0 makes a good return value.

Example

The following example comes from an application that runs in *basic* and in an *advanced* mode. The status bar for the application contains a short cut to switch modes by double clicking on a section in the status bar. The user can also switch modes with the F3 key, or from the menu. No matter which method is used, the application switches mode by invoking the *simple* method of the status bar. Therefore all the work of switching modes is done in th simple mode change event handler:

```
::method onModeChange unguarded
  use arg id, code, sb

  if sb~isSimple then do
      self~statusBarUpdates = 'off'
      self~setSimpleTasks
      self~useWizrds = .true
  end
  else do
```

```
        self~statusBarUpdates = 'on'
        self~useAdvancedTasks
        self~useWizrds = .false
    end
    return 0
```

4.7.29. connectTabEvent

```
>>--connectTabEvent(--id--,--event--+--------------+--+-------------+--)-----><
                                    +-,--methodName-+  +-,--willReply-+
```

The *connectTabEvent* method connects a specific *event* notification from a tab control with an event handling method in the Rexx dialog.

Arguments:

The arguments are:

id [required]

The resource ID of the tab control whose event notification is to be connected to a method in the Rexx dialog. May be numeric or *symbolic*.

event [required]

A single keyword indicating which event is to be connected. Case is not significant. The event keywords are:

KEYDOWN

The notification is sent when a key has been pressed while the tab control has the focus.

SELCHANGE

A new tab has been selected in the tab control. This method is called after the selection has changed.

SELCHANGING

A new tab has been selected in the tab control. This method is called before the selection is changed. The programmer can prevent the selected tab being changed at this point if the *willReply* argument is set to **`.true`**.

The selection is prevented from changing by returning **`.false`** from the event handler for this event. Returning **`.true`** allows the change. Again, **note**, that preventing the change is only possible by specifying **`.true`** for the *willReply* argument. If *willReply* is **`.false`**, the default, the reply from the event handling method is ignored.

methodName [optional]

The method name that is invoked whenever the specified notification is received from the tab control. Provide a method with a matching name. If you omit this argument, the method name is automatically generated by the ooDialog framework. The generated name will be the event keyword preceded by **on**, for example *onSelChanging*.

willReply [optional]

The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.false`**.

Return value:

The return codes are:

0

No error detected.

-1
: The resource ID could not be resolved or the event argument is incorrect.

1
: The messages was not connected correctly.

Remarks:

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

Example:

The following example invokes the method *onSelChange* whenever another tab is selected in the tab control:

```
::class MyDlgClass subclass UserDialog

::method init
  self~connectTabEvent(IDC_TAB, "SELCHANGE")
```

4.7.29.1. KeyDown Event Handler

The event handler for the key down event is invoked when the user types a key when the tab control has the focus.

The *willReply* argument in the *connectTabEvent* method determines how the event handler needs to respond to the notification.

```
::method onKeyDown unguarded
  use arg id, vKey, tabControl

  return 0
```

Arguments:

The event handling method receives 3 arguments:

id
: The numeric resource ID of the tab control sending the notification.

vKey
: The virtual key code of the key typed. The *VK* class can be used to decode this value.

arg2
: The Rexx *Tab* object representing the underlying tab control that sent the notification.

Return:

The operating system ignores the return from this notification so 0 makes a good return value.

Example

The following example lets the user navigate through the tabs by typing N for next tab, P for previous tab, HOME for the first tab and END for the last tab:

```
::method onKeyDown unguarded
  use arg id, vKey, tabControl

  lastTab = tabControl~items
```

```
select
  when vKey == .VK~N then self~onNext
  when vKey == .VK~P then self~onPrevious
  when vKey == .VK~home then self~goTo(tabControl, 0)
  when vKey == .VK~end then self~goTo(tabControl, lastTab)
  otherwise nop  -- ignore all other keys
end
-- End select

return 0
```

4.7.29.2. SelChange Event Handler

The event handler for the SELCHANGE event is invoked when the currently selected tab of the control has changed. Note that the notification is sent after the change has happened.

The *willReply* argument in the *connectTabEvent* method determines how the event handler needs to respond to the notification.

```
::method onSelChange unguarded
  use arg id, hwndFrom, tabControl

  return 0
```

Arguments:

The event handling method receives 3 arguments:

id
: The numeric resource ID of the tab control sending the notification.

hwndFrom
: The window handle of the tab control sending the notification.

tabControl
: The Rexx *Tab* object representing the underlying tab control that sent the notification.

Return:

The operating system ignores the return from this notification so 0 makes a good return value.

Example

The following example comes from a program that sometimes allows hexadecimal numbers to be used and at other times does not. When the user changes the tab to the one with index 3, and if hexadecimal numbers are currently allowed, the hexadecimal entry field is enabled, otherwise it is disabled:

```
::method onSelChange unguarded
  use arg id, hwndFrom, tabControl

  edit = self~newEdit(IDC_EDIT_HEX)

  if tabControl~selectedIndex == 3 & self~allowHexadecimal then edit~enable
  else edit~disable

  return 0
```

4.7.29.3. SelChanging Event Handler

The event handler for the SELCHANGING event is invoked when the user has selected a new tab. The notification is sent before the change has been made.

The *willReply* argument in the *connectTabEvent* method determines how the event handler needs to respond to the notification.

```
::method onSelChanging unguarded
  use arg id, hwndFrom, tabControl

  return .true
```

Arguments:

The event handling method receives 3 arguments:

id
: The numeric resource ID of the tab control sending the notification.

hwndFrom
: The window handle of the tab control sending the notification.

tabControl
: The Rexx *Tab* object representing the underlying tab control that sent the notification.

Return:

When the *willReply* argument was set to true in the connect tab event method, the event handler must return true or false. A return of true allows the tab to be changed, a return of false prevents the tab from changing.

When the *willReply* argument was set to false or SYNC in the connect tab event method, the ooDialog framwork replies to the operating system allowing the tab to change. The return from the event handler is ignored, but returning 0 is best practice.

Example

The following example validates the data entered on a page of the tab control when the user goes to switch pages. If invalid data was entered on the current page of the tab control, the user is prevented from switching to a different tab until she enters valid data. A message dialog might be put saying something like: *Your work telephone number is required. Please enter a valid telephone number before changing to a new page.*

```
::method initDialog

  self~connectTabEvent(IDC_TAB, "SELCHANGING", .true)

::method onSelChanging unguarded
  use arg id, hwndFrom, tabControl

  currentTab = tabControl~selectedIndex
  if \ self~validateTab(currentTab) then do
    -- The programmer should put up a message explaining
    -- why the page of the tab is not changing ...
    return .false
  end

  return .true
```

4.7.30. connectToolBarEvent

```
>>--connectToolBarEvent(--id--,--event--+-----------+--+-------------+--)------><
                                        +-,-mthName-+  +-,-willReply-+
```

Connects an event notification message from a toolbar control to a method in the Rexx dialog object.

Arguments:

The arguments are:

id [required]
: The resource ID of the toolbar control whose notification message is to be connected to a method in the Rexx dialog. May be numeric or *symbolic*.

event [required]
: Exactly one of the following keywords. The keyword specifies the event to be connected. Case is not significant.

BEGINADJUST	ENDADJUST	QUERYDELETE
BEGINDRAG	ENDDRAG	QUERYINSERT
CHAR	GETBUTTONINFO	RCLICK
CLICK	GETDISPINFO	RDBLCLK
CUSTHELP	GETINFOTIP	RELEASEDCAPTURE
DBLCLK	GETOBJECT	RESET
DELETINGBUTTON	HOTITEMCHANGE	RESTORE
DRAGOUT	INITCUSTOMIZE	SAVE
DRAGOVER	KEYDOWN	TOOLBARCHANGE
DROPDOWN	LDOWN	WRAPACCELERATOR
DUPACCELERATOR	MAPACCELERATOR	WRAPHOTITEM

BEGINADJUST
: Notifies the dialog that the user has begun customizing a toolbar

BEGINDRAG
: Notifies the dialog that the user has begun dragging a button in a toolbar.

CHAR
: Sent by the toolbar when a character key is pressed.

CLICK
: Sent by a toolbar control to notify the dialog that the user has clicked the left mouse button within the control.

CUSTHELP
: Notifies the dialog that the user has chosen the Help button in the Customize Toolbar dialog box.

DBLCLK
: Sent by a toolbar control to notify the dialog that the user has double-clicked the left mouse button within the control.

DELETINGBUTTON
: Sent by a toolbar when a button is about to be deleted.

DUPACCELERATOR
: Sent by a toolbar to determine whether an accelerator key can be used on two or more active toolbars.

DRAGOUT
: Sent by a toolbar control when the user clicks a button and then moves the cursor off the button.

  This notification allows an application to implement drag-and-drop functionality for toolbar buttons. When processing this notification, the application would begin the drag-and-drop operation.

DRAGOVER
: Sent by a toolbar control to determine whether a *markButton* message should be sent for a button that is being dragged over.

DROPDOWN
: Sent by a toolbar control when the user clicks a dropdown button.

  Dropdown buttons can be plain (DROPDOWN style), display an arrow next to the button image (WHOLEDROPDOWN style), or display an arrow that is separated from the image (DRAWDDARROWS extended style). If a separated arrow is used, DROPDOWN is sent only if the user clicks the arrow portion of the button. If the user clicks the main part of the button, a button CLICKED event notification is sent, just as with a standard button. For the other two styles of dropdown button, DROPDOWN is sent when the user clicks any part of the button.

ENDDRAG
: Notifies the dialog that the user has stopped dragging a button in a toolbar.

ENDADJUST
: Notifies the dialog that the user has ended customizing a toolbar.

GETBUTTONINFO
: Retrieves toolbar customization information and notifies the dialog of any changes being made to the toolbar.

GETDISPINFO
: Retrieves display information for a toolbar item.

GETINFOTIP
: Retrieves infotip information for a toolbar item.

GETOBJECT
: Sent by a toolbar control that uses the REGISTERDROP style to request a drop target object when the pointer passes over one of its buttons.

HOTITEMCHANGE
: Sent by a toolbar control when the hot (highlighted) item changes.

INITCUSTOMIZE
: Notifies the dialog that customizing has started.

KEYDOWN
: Sent by a toolbar control when the control has the keyboard focus and the user presses a key.

LDOWN
: Notifies the dialog that the left mouse button has been pressed.

MAPACCELERATOR
: Requests the index of the button in the toolbar corresponding to the specified accelerator character.

QUERYDELETE
: Notifies the dialog whether a button may be deleted from a toolbar while the user is customizing the toolbar. The event handler can accept or refuse the deletion.

QUERYINSERT
: Notifies the dialog that a button may be inserted to the left of the specified button while the user is customizing a toolbar. The event handler can accept or refuse the insertion.

RCLICK
: Sent by a toolbar control to notify the dialog that the user has clicked the right mouse button within the control.

RDBLCLK
: Sent by a toolbar control to notify the dialog that the user has double-clicked the right mouse button within the control.

RELEASEDCAPTURE
: Notifies the dialog that the toolbar is releasing the mouse capture.

RESET
: Notifies the dialog that the user has reset the content of the Customize Toolbar dialog box.

RESTORE
: Notifies the dialog that a toolbar is in the process of being restored.

SAVE
: Notifies the dialog that a toolbar is in the process of being saved.

TOOLBARCHANGE
: Notifies the dialog that the user has customized a toolbar.

WRAPACCELERATOR
: Requests the index of the button in one or more toolbars corresponding to the specified accelerator character.

WRAPHOTITEM
: Notifies an application with two or more toolbars that the hot item is about to change.

methodName [optional]
: The name of the method that is to be invoked whenever the specified event happens. The programmer defines this method. If this argument is omitted, a method name is automatically generated that consists of the event keyword preceded by **`on`**. For instance, **`onBeginAdjust`**. The method name can not be the empty string. The empty string is treated as an omitted argument.

willReply [optional]
: The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.true`**.

For those event notifications that the operating system expects a return value, the *willReply* argument is ignored. For these events, the event handler must always return a value of the correct type. I.e., if the operating system expects a reply of true or false, the event handler must always return true or false. The events in the following list fall into this category:

CHAR	GETBUTTONINFO	MAPACCELERATOR
CLICK	GETDISPINFO	QUERYDELETE
KEYDOWN	GETINFOTIP	QUERYINSERT
RCLICK	GETOBJECT	RESTORE
RDBLCLK	HOTITEMCHANGE	
DROPDOWN	INITCUSTOMIZE	

Return value:

Returns **`.true`** if the event was connected correctly, otherwise **`.false`**.

Remarks:

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

Details:

Syntax errors are raised when incorrect usage is detected, including the use of an invalid symbolic ID or an unrecognized event keyword.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if the connected event happens.

The underlying dialog receives the TBN_* messages as the notifications for the toolbar events.

4.7.30.1. General ToolBar Event Handler

A number of the toolbar events are processed internally by ooDialog in the same manner. The event handlers for these events are sent the same arguments and are essentially coded the same way. This *general* event handler is described here for the BEGINADJUST, CUSTHELP, ENDADJUST, and TOOLBARCHANGE events. The description for event keyword in the *connectStatusBarEvent* documentation details when each event will invoke this event handler.

The *willReply* argument in the *connectToolBarEvent* method determines how the event handler needs to respond to the notification.

```
::method onToolBarEvent unguarded
  use arg id, notifyCode, toolBar

  return .true
```

Arguments:

The event handling method receives 3 arguments:

id

The numeric resource ID of the statusbar sending the notification.

notifyCode

The numeric notification code of the event that caused the notification to be sent. Each dialog control has its own specific notification codes.

toolBar
: The Rexx toolbar object that represents the underlying dialog control that sent the notification. It is possible this will be the **.nil** object if some error happened, but this is very unlikely.

Return:

The operating system ignores the return value for these events. 0 makes a good value to return.

Example

The following example

```
```

4.7.30.2. General ToolBar Mouse Click Event Handler

A number of the toolbar events related to mouse clicks are processed internally by ooDialog in the same manner. The event handlers for these events are sent the same arguments and are essentially coded the same way. This *general* click event handler is described here for the CLICK, DBLCLK, RCLICK, and RDBLCLK events. The description for event keyword in the *connectToolBarEvent* documentation details when each event will invoke this event handler.

The event handler must always reply either true or false to respond to these notification.

```
::method onToolBarClick unguarded
  use arg id, nCode, buttonID, pt, toolBar

  return .true
```

Arguments:

The event handling method receives 5 arguments:

id
: The numeric resource ID of the toolbar sending the notification.

nCode
: The numeric notification code of the event that caused the notification to be sent. Each dialog control has its own specific notification codes.

index
: The resource ID of the button that was clicked, or 0 if the click was not on a button in the toolbar.

pt
: A *Point* object that specifies the location of the click. Not that MSDN says this point is in screen *coordinates*, but from experimentation it appears to actually be in client coordinates.

toolBar
: The Rexx toolbar object that represents the underlying dialog control that sent the notification. It is possible this will be the **.nil** object if some error happened, but this is very unlikely.

Return:

The event handler must return true or false. Return true to indicate that the mouse click was handled and suppress default processing by the operating system. Return false to allow default processing of the click.

Example

The following example has a toolbar with 3 buttons. The application allows the user to right click on the buttons to perform an extended action of the buttons purpose. For instance, one button is a File Open button. A regular click opens a file in the default encoding, a right click allows the user to specify the encoding to open the file:

```
::method onRightClick unguarded
  use arg id, nCode, buttonID, pt, toolBar

  if buttonID <> 0 then do
    self~buttonExtendedAction(buttonID, toolBar)
  end

  return .true
```

4.7.31. connectToolTipEvent

```
>>--connectToolTipEvent(--id--,--event--+-----------+--+-------------+--)------><
                                        +-,-mthName-+  +-,-willReply-+
```

The *connectToolTipEvent* method connects an *event* notification message from a *ToolTip* control to a method in the Rexx dialog.

Arguments:

The arguments are:

id [required]

The resource ID of the tool tip control whose notification message is to be connected to a method in the Rexx dialog. May be numeric or *symbolic*.

event [required]

Exactly one of the following keywords. The keyword specifies the event to be connected. Case is not significant.

LINKCLICK	POP
NEEDTEXT	SHOW

LINKCLICK

Requires Common Control *Library* version 6.0 or later.

This notification is sent when a text link inside a balloon ToolTip is clicked. MSDN provides this example of when this notification is sent. Assume that the balloon ToolTip contains the following text:

```
         "This is a <A>link</A>".
```

When *link* is clicked, the LINKCLICK notification is sent.

NEEDTEXT

This notification is sent when the ToolTip control needs to retrieve the text used to display a ToolTip window.

POP

This notification is sent when a ToolTip is about to be hidden.

SHOW
: This notification is sent when a ToolTip control is about to be displayed.

methodName [optional]
: The name of the method that is to be invoked whenever the specified event happens. The programmer defines this method. If this argument is omitted, a method name is automatically generated that consists of the event keyword preceded by **on**. For instance, **onNeedText**. The method name can not be the empty string. The empty string is treated as an omitted argument.

willReply [optional]
: The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **.true**.

  However, for the SHOW and NEEDTEXT events, the *willReply* argument is ignored. The event handler for those events must always return a value.

Return value:

Returns **.true** if the event was connected correctly, otherwise **.false**.

Remarks:

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

Details:

Syntax errors are raised when incorrect usage is detected, including the use of an invalid symbolic ID or an unrecognized event keyword.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if the connected event happens.

The underlying dialog receives the TTN_* messages as the notifications for the tool tip events.

4.7.31.1. LinkClick Event Handler

The event handler for the LINKCLICK event is invoked when the text link inside a Balloon ToolTip is clicked.

The *willReply* argument in the *connectToolTipEvent* method determines how the event handler needs to respond to the notification.

```
::method onLinkClick unguarded
  use arg rxToolID, rxToolTip

  return 0
```

Arguments:

The event handling method receives two arguments:

rxToolID
: The Rexx object used as the second part of the tool *identification* the operating system uses. This can be the Rexx dialog control object or the numeric ID that identifies the tool.

rxToolTip
: The Rexx tool tip object.

Return:

The actual value of the return has no meaning. 0 makes a good return.

4.7.31.2. NeedText Event Handler

The event handler for the NEEDTEXT event is invoked when the ToolTip is about to be displayed and the ToolTip needs the text to be displayed. When a tool is *added* to a ToolTip, the programmer normally supplies a literal string to be used as the text. However, the ToolTip provides a special value which essentially tells the ToolTip to call back to the application when it needs the text to be displayed. The ToolTip calls back by sending the NEEDTEXT notification.

The programmer must return a string to be used for the text and the interpreter waits for this return. The returned text is passed back to the ToolTip.

```
::method onNeedText unguarded
  use arg toolID, toolTip, info

  return .true
```

Arguments:

The event handling method receives 3 arguments:

toolID

The Rexx object used as the second part of the tool *identification* the operating system uses. This can be the Rexx dialog control object or the numeric ID that identifies the tool.

toolTip

The Rexx tool tip object.

info

A **`Directory`** object whose indexes are used to convey information concerning the event notification to the event handler and with which the event handler returns the requested text. On invocation of the event handler, the **`Directory`** object will have the following indexes with the corresponding information:

TEXT

The event handler sets this index to the text to be used for the displayed ToolTip. If this index is set to the empty string, then no ToolTip is displayed. The length of the returned text must be less than 1024 characters. The index is set to the empty string when the *info* argument is passed to the event handler.

USERDATA

This index is set to the *user data* specified for the tool. For example, in the *addTool*, *addToolEx*, or *addToolRect* methods. If there is no user data, the index is set to the **`.nil`** object.

FLAGS

Zero or more blank separated keywords indicating the tool's flags. The possible flag keywords are:

ABSOLUTE	PARSELINKS	TRACK
CENTERTIP	RTLREADING	TRANSPARENT
IDISHWND	SUBCLASS	

ABSOLUTE
: Positions the ToolTip window at the exact same coordinates specified by the *trackPosition* method. Without this flag the ToolTip control chooses where to display the ToolTip window based on the coordinates specified, which places the ToolTip close to the tool. This flag must be used with the TRACK flag.

CENTERTIP
: Indicates that the ToolTip window is centered below the tool.

IDISHWND
: Indicates that the ID part of the tool *identification* is the window handle to the tool. If this flag is not set, the ID part is the tool's identification number.

PARSELINKS
: Indicates that links in the ToolTip text should be parsed.

  Only available with Common Control *Library* version 6.0 or later.

RTLREADING
: Indicates that the ToolTip text will be displayed in the opposite direction to the text in the parent window.

SUBCLASS
: Indicates that the ToolTip control should subclass the tool's window to intercept messages, such as WM_MOUSEMOVE. If this flag is not set, the mouse messages must be forwarded to the ToolTip control.

TRACK
: Positions the ToolTip window next to the tool to which it corresponds and moves the window according to coordinates supplied by the *trackPosition* method.

TRANSPARENT
: Causes the ToolTip control to forward mouse event messages to the parent window. This is limited to mouse events that occur within the bounds of the ToolTip window.

Return:

The event handler must return true or false. When false is returned, the ToolTip continues to generate NEEDTEXT notifications each time it is about to display the tool tip window. If true is returned, the ToolTip control stores the information and will not request it again.

Example

The following example shows a simple NEEDTEXT event handler that sets the tool tip text when the mouse hovers over a push button. True is returned to indicate that the ToolTip should store the text for that tool and need not ask for it again:

```
::method onNeedText unguarded
    use arg id, toolTip, info

    info~text = 'Press Test to execute the regression test suite ...'
    return .true
```

4.7.31.3. Pop Event Handler

The event handler for the POP event is invoked when the ToolTip is about to hide a displayed tip.

The *willReply* argument in the *connectToolTipEvent* method determines how the event handler needs to respond to the notification.

```
::method onPop unguarded
  use arg rxToolID, rxToolTip

  return 0
```

Arguments:

The event handling method receives two arguments:

rxToolID

The Rexx object used as the second part of the tool *identification* the operating system uses. This can be the Rexx dialog control object or the numeric ID that identifies the tool.

rxToolTip

The Rexx tool tip object.

Return:

The actual value of the return has no meaning. 0 makes a good return value.

4.7.31.4. Show Event Handler

```
::method onShow unguarded
  use arg rxToolID, rxToolTip

  return trueOrFalse
```

The event handler for the SHOW event is invoked when the ToolTip is about to display its window.

The programmer must always return a value from the event handler and the interpreter waits for this return. The *willReply* argument of the *connectToolTipEvent* method is ignored for this event. If the programmer does not intend to return a value from the event handler, the event should not be connected.

Arguments:

The event handling method receives two arguments:

rxToolID

The Rexx object used as the second part of the tool *identification* the operating system uses. This can be the Rexx dialog control object or the numeric ID that identifies the tool.

rxToolTip

The Rexx tool tip object.

Return:

To display the ToolTip in its default position, return false. To customize the position, reposition the window and return true.

Remarks:

Writing an event handler for the SHOW event gives the programmer the opportunity to customize the position of the ToolTip. Within the event handler, calculate the desired position of the ToolTip, then use the *setWindowPos* to place the ToolTip window in that position. The event handler needs to return true to inform the ToolTip that custom positioning has been performed. Note that MSDN says that only the position of the window should be changed and that trying to change the size of the window will produce unpredictable results.

A ToolTip's window rectangle is somewhat larger than its text display rectangle, and its origin is offset up and to the left. If the application needs to accurately position the text display rectangle of a ToolTip, use the *adjustRect* method. This converts a text display rectangle into the corresponding ToolTip window rectangle and vice versa.

Example:

The following example uses the ToolTip ID to determine which ToolTip tool is about to display. It then calculates the position the ToolTip should be shown at, and uses *setWindowPos* to position the window at the point calculated. Finally, true is returned, so the ToolTip knows not to position itself at its default position:

```
::method onShow unguarded
  expose r1 r2 r3 r4
  use arg toolID, toolTip

  select
    when toolID == .constDir[IDTOOL_DLG_RECT1] then pos = .Point~new(r1~left, r1~top)
    when toolID == .constDir[IDTOOL_DLG_RECT2] then pos = .Point~new(r2~left, r2~top)
    when toolID == .constDir[IDTOOL_DLG_RECT3] then pos = .Point~new(r3~left, r3~top)
    when toolID == .constDir[IDTOOL_DLG_RECT4] then pos = .Point~new(r4~left, r4~top)
    otherwise pos = .Point~new
  end
  -- End select

  self~client2screen(pos)
  toolTip~setWindowPos(TOPMOST, pos, .Size~new(0, 0), "NOACTIVATE NOSIZE NOZORDER")

  return .true
```

4.7.32. connectTrackBarEvent

```
>>--connectTrackBarEvent(--id--,--event--+----------+--+-------------+--)------><
                                          +-,--mName-+  +-,-willReply-+
```

The *connectTrackBarEvent* method connects a particular trackbar POSITION event notification to a method in the Rexx dialog.

Take Note

The method can only be called after the underlying trackbar has been created. A good location for this connection is the *initDialog* method.

Arguments:

The arguments are:

id [required]

The resource ID of the trackbar control whose notification is to be connected.

event [required]
: Exactly one of the following keywords that specifies the event to be connected. Case is not significant:

  UP
  : The Up or right key has been pressed.

  DOWN
  : The Down or left key has been pressed.

  TOP
  : The Home key has been pressed.

  BOTTOM
  : The End key has been pressed.

  PAGEUP
  : The PgUp key has been pressed.

  PAGEDOWN
  : The PgDn key has been pressed.

  DRAG
  : The thumb of the trackbar has been moved.

  POSITION
  : The left mouse button has been released, following a DRAG notification.

  ENDTRACK
  : The trackbar movement is completed, that is, the appropriate key or mouse button has been released.

mName [optional]
: The name of the method to invoke whenever the specified notification is received the dialog. Provide a method with a matching name. If this argument is omitted, the ooDialog framework generates a method name automatically. The name will the event keyword preceded by **on**. For example, *onDrag*.

willReply [optional]
: The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.false`**.

Return value:

The return codes are:

0
: No error detected.

-1
: The resource ID could not be resolved or the event argument is incorrect.

1
: The messages was not connected correctly.

Example:

The following example connects the POSITION event (the mouse button is released after dragging) with method *posSet*, which extracts the new trackbar position from the notification arguments and displays it together with the event type for POSITION, which should be 4:

```
::class ExampleDialog subclass UserDialog

::method initDialog

  self~connectTrackBarEvent("MYSLIDER", "POSITION", onPosition, 'SYNC')

::method onPosition unguarded
  use arg posInfo, hwnd, trackBar

  pos = .DlgUtil~hiWord(posInfo)

  say 'TrackBar new position:          ' pos
  say "Verify event code (should be 4):" .DlgUtil~loWord(posInfo)

  return 0

/*  Output might be:

TrackBar new position:           80
Verify event code (should be 4): 4
TrackBar new position:           0
Verify event code (should be 4): 4
TrackBar new position:           29
Verify event code (should be 4): 4

*/
```

4.7.32.1. TrackBar Move Event Handler

The event handler for the Move event is invoked when the user moves the moves the slider in a trackbar.

The *willReply* argument in the *connectTrackBarEvent* method determines how the event handler needs to respond to the notification.

```
::method onPage unguarded
  use arg posInfo, hwnd, trackBar

  return 0
```

Arguments:

The event handling method receives 3 arguments:

posInfo
: The *posInfo* argument contains the move event code in the low word of the argument. For the POSITION and the DRAG move events only, the high word contains the position of the thumb box of the slider in the trackbar. For all the other events, the high word will be zero.

hwnd
: The *hwnd* argument is the window handle of the trackbar.

trackBar
: The Rexx trackbar object that represents the underlying trackbar that sent the notification.

Return:

The operating system ignores the return from this notification so 0 makes a good return value.

Remarks:

For the event notifications where the high word of *posInfo* is zero, the position can be gotten directly from the *trackBar* object sent as the third argument

The *TrackBar* class provides some *constant* values as a convenience to help decode the numeric move event code:

```
::method onMove unguarded
  use arg posInfo, hwnd, trackBar

  code = .DlgUtil~lowWord(posInfo)

  if code == trackBar~UP then do
    ...
  end
  else if code == trackBar~DOWN then do
    ...
  end
  else if code == trackBar~PAGEUP then do
    ...
  end
  --- and so on ...
```

Example

The following example sets a static text label to the new position of a trackbar each time a move event occurs.

```
    ::method onEndTrack unguarded
    expose tbLabels
    use arg eventInfo, hwndTrackBar, trackBar

    -- Get the resource ID of the trackBar and use that as an index into the
    -- table of the labels.
    id = trackBar~id
    tbLabels[id]~setText(trackBar~pos)
```

4.7.33. connectTreeViewEvent

```
>>-connectTreeViewEvent(--id--,--event--+-------------+--+-------------+--)----><
                                        +-,--mthName--+  +-,-willReply-+
```

The *connectTreeViewEvent* method connects a method in a Rexx dialog to a particular tree view *event*.

Arguments:

The arguments are:

id

The resource ID of the tree view control. This can be the numeric or symbolic ID.

event

A keyword indicating which event is to be connected. Case is not significant. The keyword must be one of the following:

BEGINDRAG	ENDEDIT	KEYDOWNEX
BEGINEDIT	EXPANDED	SELECTCHANGED
BEGINRDRAG	EXPANDING	SELECTCHANGING
DEFAULTEDIT	GETINFOTIP	
DELETE	KEYDOWN	

BEGINDRAG

A drag-and-drop operation using the left mouse button was initiated. The documentation for the *defTreeDragHandler*() method contains further information on how to implement a drag-and-drop handler. Not that if the tree-view control is give the *NODRAG* style, the BEGINDRAG notification is not sent.

BEGINEDIT

Editing a label has been started. Do not connect this event if you are using the DEFAULTEDIT keyword. The results are undefined. The tree-view must have the *EDIT* style for this notification to be sent.

The event notification for this event has been enhanced since the original ooDialog implementation. To use the enhanced event notification, the *willReply* argument must be used. The value of the argument, true or false, does not matter. If *willReply* is omitted, the old style notification is used. The documentation for the *BEGINEDIT* event handler explains the difference between the two types of notifications.

BEGINRDRAG

A drag-and-drop operation involving the right mouse button was initiated. The documentation for the *defTreeDragHandler*() method contains further information on how to implement a drag-and-drop handler. handler.

DEFAULTEDIT

This keyword is used to activate the internal handling of the tree-view label editing operation. With this keyword, the ooDialog framework internally handles the BEGINEDIT and ENDEDIT notifications. The tree-view must have the *EDIT* style for the BEGINEDIT / ENDEDIT notification to be sent. While using the DEFAULTEDIT connection may seem easier than using the BEGINEDIT / ENDEDIT connections, it does not provide the same flexibility as using the BEGINEDIT / ENDEDIT connections.

When you specify this event connection, omit the *methodName* argument, the arugment is ignored. Do not connect either the BEGINEDIT or ENDEDIT events when also using the DEFAULTEDIT connection. The result is undefined.

DELETE

An item has been deleted.

ENDEDIT

Label editing has ended. Do not connect this event if you are using the DEFAULTEDIT keyword. The results are undefined. The tree-view must have the *EDIT* style for this notification to be sent.

The event notification for this event has been enhanced since the original ooDialog implementation. To use the enhanced event notification, the *willReply* argument must be used. The value of the argument, true or false, does not matter. If *willReply* is omitted, the old style notification is used. The documentation for the *ENDEDIT* event handler explains the difference between the two types of notifications.

EXPANDED
: An item has expanded or collapsed. This notification is sent after the item expanded or collapsed.

EXPANDING
: An item is about to expand or collapse. This notification is sent before the item has expanded or collapsed.

GETINFOTIP
: The tree-view control is requesting text to display an info tip. The notification is only sent when the tree-view control has the *INFOTIP* style.

KEYDOWN
: A key was pressed inside the tree view. This notification is not sent while a label is being edited.

KEYDOWNEX
: A key was pressed inside the tree view. This notification is not sent while a label is being edited.

  This event is exactly the same as the KEYDOWN event. Except, when this keyword is used, the ooDialog framework sends a different set of arguments to the event handler. The additional arguments provide more information to the programmer than the arguments sent when the KEYDOWN keyword is used. The two keywords are needed to provide backwards compatibility.

SELCHANGED
: Another item was selected. This notification is sent after the selection was changed.

SELCHANGING
: Another item is about to be selected. This notification is sent before the selection has changed.

mthName
: The name of the event handling method. This method is invoked each time the specified event occurs for the tree view control. If you omit this argument, the method name is generated for you. This name will be the event keyword, preceded by **on**. For example: *onExpanded*. The method name can not be the empty string.

willReply [optional]
: The *willReply* argument controls how the interpreter invokes the event handler for the connected event. The default if *willReply* is omitted is **`.false`**.

  The operating system expects a return value from the GETINFOTIP notification. When that event is connected the *willReply* argument is ignored. The event handler for GETINFOTIP must always return a string value.

Return value:

0
: No error detected.

-1
: The resource ID was symbolic and it could not be resolved, or the event keyword is not correct.

1
: Some other error and the message was not connected correctly. This may indicate the message table is full, or the interpreter is out of usable memory.

Remarks:

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

Note: The original implementation of ooDialog was limited in the arguments sent to the event handlers for many event notifications. In many cases, the Windows notification provides more information than was provided to the event handlers in the original implementation. Many of the event notifications for the tree-view have been enhanced to provide the event handlers with more complete information. To maintain backwards compatibility, if the *willReply* argument is omitted, the old event notification implementation is used. To use the enhanced implementation, the programmer simply includes the *willReply* argument. Whether the argument is false or true does not matter. It is the process of including the argument that signals the programmer wants to use the enhanced notification.

However, the *willReply* argument is ignored for the GETINFOTIP event. For this event, the interpreter always waits for the reply. If the programmer does not wish to return a value from the event handler for the GETINFOTIP, then he should not connect that event.

For each tree-view event keyword, there is a section describing the details of the event handler for that keyword. These sections immediately follow this section.

Details:

Syntax errors are raised when incorrect usage is detected.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if any command events happen.

In Windows itself, tree view event notifications are sent to the parent dialog using the WM_NOTIFY message.

Example:

The following example connects the selection-changed event for the tree-view with the symbolic ID of IDC_TV_FILES with the method newTreeSelection in the Rexx dialog. The event handling method displays the text of the new selection:

```
::class 'FileDlg' subclass UserDialog

::method init
  self~connectTreeViewEvent(IDC_TV_FILES, "SELCHANGED", "newTreeSelection", .true)

::method newTreeSelection unguarded
  use arg id, hItemOld, userOld, hItemNew, userNew, action, rxTv
  textNew = rxTv~itemText(hItemNew)
  say "The label of the new selection is:" text

  return 0
```

4.7.33.1. BeginDrag / BeginRDrag Event Handler

The event handler for the BEGINDRAG event is invoked when a drag-and-drop operation involving the left mouse button is being initiated in the tree-view. The event handler for the BEGINRDRAG event is invoked when a drag-and-drop operation involving the right mouse button is being initiated in the tree-view. The arguments sent to, and the behaviour of, both event handlers is identical. Both event handlers are described in this single section.

Both of these event notifications have been enhanced since the original ooDialog implementation. If the *willReply* argument to the *connectTreeViewEvent* method is omitted the old implementation is used. If the argument is not omitted, the new implementation is used. In addition, ooDialog provides a default handler for these events. If the *mthName* argument is specified as *defTreeDragHandler*, then this default event handler is used. How the event handlers work is described separately below:

New event handler description:

The *willReply* argument in the *connectTreeViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onBeginDrag unguarded
  use arg id, hItem, pos, treeView, itemData

  return 0
```

Arguments:

The event handling method receives 5 arguments:

id

The resource id of the tree-view sending the notification.

hItem

The tree-view item handle that is being dragged

pos

The position, in client *coordinates*, where the drag started. *pos* is a *Point* object.

treeView

The Rexx tree-view object whose underlying tree-view control has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newTreeView* method.

itemData

The item *data* object assigned to the tree-view item being dragged. If no item data has been assigned, this argument will be the **`.nil`** object.

Return:

The operating system ignores the returned value, so any value can be used. 0 makes a good return.

Old event handler description:

The old style event notification is used when the programmer omits the *willReply* argument in the invocation of the *connectListViewEvent* method. The return from the event handler is completely ignored, the interpreter does not wait for this return.

```

  ::method onBeginDrag unguarded
    use arg id, hItem, where, treeView
```

Arguments:

The event handling method receives 3 arguments:

id
: The resource id of the tree-view sending the notification.

hItem
: The handle to the tree-view item being dragged

where
: Specifies the postion of the cursor when the drag operation was initiated as a 2 word string. The first word is the x position, the second word is the y postion. The position is expressed in client *coordinates*.

treeView
: The Rexx tree-view object whose underlying tree-view control has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newTreeView* method.

Return:

The return value is ignored by the operating system. 0 makes a good return value.

Example:

```
::method onBeginDrag unguarded
  use arg id, item, where, treeView
  say "Item with handle" item "is in drag-and-drop mode"
  parse var where x y
  say "The drag operation started at point ("x","y")"

  return 0
```

defTreeDragHandler event handler description:

If the *mthName* argument for the BEGINDRAG or BEGINRDRAG event connections is *defTreeDragHandler*, then a method supplied by the ooDialog framework is connected. The programmer does not take any other action. Since the ooDialog framework is handling things internally, the *willReply* argument is ignored when *defTreeDragHandler* is used.

4.7.33.2. BeginEdit Event Handler

The event handler for the BEGINEDIT event is invoked when the user begins a label editing operation on an item of the tree-view. When label editing begins, an edit control is created by the operating system, but not positioned or displayed. Before it is displayed, the tree-view control sends a BEGINEDIT notification message. A label editing operation is only available when the tree-view has the *EDIT* style.

In general, the programmer would connect both the BEGINEDIT and *ENDEDIT* notifications. Both of these event notifications have been enhanced since the original ooDialog implementation. If the *willReply* argument to the *connectTreeViewEvent* method is omitted the old implementation is used. If the argument is not omitted, the new implementation is used. How the two event handlers work is described separately.

New event handler description:

The *willReply* argument in the *connectTreeViewEvent* method determines how the event handler needs to respond to the notification.

```

  ::method onBeginEdit unguarded
```

```
    use arg id, hItem, editCtrl, treeViewCtrl, itemData

    return zz
```

Arguments:

The event handling method receives 5 arguments:

id

The resource id of the tree-view sending the notification.

hItem

The tree-view item handle whose label is about to be edited.

editCtrl

The Rexx edit control object used for the editing operation. The programmer can customize the editing operation by using the methods of the *Edit* class.

Note that this object is only valid during the editing operation. When the user ends the editing, the operating system destroys the underlying edit control and the Rexx object is no longer valid. Each time the user starts a new editing operation, the operating system creates a new edit control.

treeViewCtrl

The Rexx tree-view object whose underlying tree-view control has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newTreeView* method. Unlike the *editCtrl* object, this object is valid as long as the dialog is executing.

itemData

The item *data* object assigned to the tree-view item whose label is about to be edited. If no item data has been assigned, this argument will be the **`.nil`** object.

Return:

When the programmer used true for the *willReply* argument, the event handler must return true or false. To allow the editing operation to begin, return true. To cancel the editing operation, return false. Returning a value from the event handler gives the programmer the option determining if the label for the specific tree-view item should or should not be edited.

IF the programmer used false or SYNC for the *willReply* argument, the ooDialog framework always replies to the operaring system to allow the editing operation to begin.

Example

The following example shows a possible event handler for the BEGINEDIT event. It uses the *hItem* argument to determine which item is about the have its label edited, and checks that editing is allowed for that item. If it is, it returns true to allow the operation. If it is not, it returns false to cancel the operation and puts up a message box to inform the user. Note that the *itemData* argument is assigned to the *rec* variable in this example, just to make it clear that the argument sent to the event handler is the item data object. In this particular program the item data object is a record object:

```
::method onBeginEdit unguarded
  use arg id, hItem, editCtrl, treeViewCtrl, itemData

  rec = itemData
  if rec~isEditable then return .true

  reply .false
```

```
msg = "The record for" rec~FirstName rec~LastName 'can not be changed.'
title = "Label Edit Error"
j = MessageDialog(msg, self~hwnd, title, , "WARNING")

return
```

Old event handler description:

The old style event notification is used when the programmer omits the *willReply* argument in the invocation of the *connectTreeViewEvent* method. The return from the event handler is completely ignored, the interpreter does not wait for this return.

```
::method onBeginEdit unguarded
  use arg id, hItem, treeView
```

Arguments:

The event handling method receives 2 arguments:

id
: The resource id of the tree-view sending the notification.

hItem
: The handle to the tree-view item whose label is about to be edited.

treeView
: The Rexx tree-view object whose underlying tree-view control has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newTreeView* method.

Return:

Returning, or not returning, a value has no meaning.

Remarks

Connecting this event and not using the *willReply* argument does not make much sense. The event handler really serves no purpose.

4.7.33.3. DefaultEdit Event Handler

When the programmer specifies the DEFAULTEDIT event keyword for the *event* argument in the *connectTreeViewEvent* method, the ooDialog framework handles both the BEGINEDIT and ENDEDIT events internally. These events are only generated when the tree-view has the *EDIT* style.

The internal handling of the events changes the label of the item if the user does not cancel the editing operation. If the user cancels the editing operation, the original lable is left unchanged. Allowing ooDialog to handle these events is less work for the programmer, but it is also inflexible.

4.7.33.4. Delete Event Handler

The event handler for the DELETE event is invoked when the user deletes an item in the tree-view.

This event notification has been enhanced since the original ooDialog implementation. If the *willReply* argument to the *connectTreeViewEvent* method is omitted the old implementation is used. If the argument is not omitted, the new implementation is used. How the two event handlers work is described separately below.

New event handler description:

The *willReply* argument in the *connectTreeViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onDelete unguarded
  use arg, id, hItem, rxTv, itemData

  return 0
```

Arguments:

The event handling method receives 4 arguments:

id

The resource id of the tree-view sending the notification.

hItem

The item handle of the item being deleted.

rxTv

The Rexx tree-view object whose underlying tree-view control has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newTreeView* method.

itemData

The item *data* object assigned to the tree-view item that is being deleted. If no item data has been assigned, this argument will be the **`.nil`** object.

Return:

The operating system ignores the return from this notification. 0 makes a good return for cases where the operating system ignores the return value from an event notification.

Old event handler description:

The old style event notification is used when the programmer omits the *willReply* argument in the invocation of the *connectTreeViewEvent* method. The return from the event handler is completely ignored, the interpreter does not wait for this return.

```
::method onDelete unguarded
  use arg id, useLess, rxTv
```

Arguments:

The event handling method receives 3 arguments:

id

The resource id of the tree-view sending the notification.

useLess

This argument is a number that has no meaning within the Rexx code. The old implementation of this event notification sent the numeric value of a pointer to memory to the event handler. However, there is nothing that can be done with the number in a Rexx program.

rxTv

The Rexx tree-view object whose underlying tree-view control has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newTreeView* method.

Return:

The interpreter does not wait for the return from the event handler, so the return has no meaning. Good practice would be to return a value anyway. 0 makes a good return value.

4.7.33.5. EndEdit Event Handler

The event handler for the ENDEDIT event is invoked when the user finishes a label editing operation on an item of the tree-view. A label editing operation is only available when the tree-view has the *EDIT* style.

In general, the programmer would connect both the *BEGINEDIT* and ENDEDIT notifications. Both of these event notifications have been enhanced since the original ooDialog implementation. If the *willReply* argument to the *connectTreeViewEvent* method is omitted the old implementation is used. If the argument is not omitted, the new implementation is used. How the two event handlers work is described separately below.

New event handler description:

The *willReply* argument in the *connectTreeViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onEndEdit unguarded
  use arg id, hItem, text, treeViewCtrl, itemData

  return trueOrFalse
```

Arguments:

The event handling method receives 5 arguments:

id

The resource id of the tree-view sending the notification.

hItem

The item handle of the tree-view item that was edited by the user.

text

If the user canceled the edit operation then the *text* argument will be the .nil object. If the edit operation was not canceled then this argument will be the text the user entered.

treeViewCtrl

The Rexx tree-view object whose underlying tree-view control has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newTreeView* method.

itemData

The item *data* object assigned to the tree-view item whose label was edited. If no item data has been assigned, this argument will be the **`.nil`** object.

Return:

When the programmer used true for the *willReply* argument, the event handler must return true or false. To accept the edited text, return true. To disallow the change to the label, return false. If, the edit operation was canceled by the user, the operating system ignores the return from the event handler. Returning a value from the event handler gives the programmer the option of determining if the new label for the specific tree-view item is acceptable.

Example

The following example checks the new text entered by the user. The label for each tree-view item is a part number. If the user's text is not a valid part number, the new text is rejected by returning false:

```
::method onEndEdit unguarded
  use arg id, hItem, text, treeViewCtrl, itemData

  if text == .nil then return .false

  if self~isValidPart(text) then do
    reply .true

    rec = itemData
    oldPartNo = rec~partNo
    rec~partNo = text
    self~updateRecord(oldPartNo, rec)

    return
  end

  reply .false

  msg = text "is not a valid part number." || .endOfLine~copies(2 || -
        "The change is rejected."

  title = "Label Editing Error"
  j = MessageDialog(msg, self~hwnd, title, , "WARNING")

  return
```

Old event handler description:

The old style event notification is used when the programmer omits the *willReply* argument in the invocation of the *connectTreeViewEvent* method. The return from the event handler is completely ignored, the interpreter does not wait for this return. If the user canceled the edit operation, the label will be unchanged. If the user did not cancel the edit operation, the label of the item is changed to the text the user entered.

```
::method onEndEdit unguarded
  use arg id, hItem, maybeText
```

Arguments:

The event handling method receives 3 arguments:

id

The resource id of the tree-view sending the notification.

hItem

The handle of the tree-view item that was edited.

text [optional]

If the user canceled the edit operation, the *text* argument is omitted. If the user did not cancel, then the *text* argument is the text the user entered.

Return:

Returning, or not returning, a value has no meaning. The interpreter does not wait for the return and its value, if any is discarded.

Remarks

When the user does not cancel the edit operation, the operating system automatically changes the label of the item to what the user entered. To prevent this behavior, the programmer needs to use the new style event handler by using the *willReply* argument to the *connectTreeViewEvent* method.

4.7.33.6. Expanded Event Handler

The event handler for the EXPANDED event is invoked after a tree-view item has been expanded or collapsed.

The *willReply* argument in the *connectTreeViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onExpanded unguarded
  use arg id, hItem, what, extra, treeView, notifyCode

  return 0
```

Arguments:

The event handling method receives 6 arguments:

id

The resource ID of the tree-view control sending the notification.

hItem

The handle of the tree-view item that expanded or collapsed.

what

A string that indicates whether the item was expanded or collapsed. The string will be either *EXPANDED* or *COLLAPSED*.

extra

The *extra* arugment will usually be the empty string. The Microsoft documentation seems to indicate that when an item is expanded, the action may be expanded partial and when the action is collapsed, it may be collapsed and reset. If expanded partial is detected, the *extra* argument will be *PARTIAL*. If collapse and reset is detected, the *extra* argument will be *RESET*. However, in testing, neither of these 2 actions were ever detected and the Microsoft documentation is unclear here. It may be that the *extra* argument will always be the empty string.

treeView

The Rexx tree-view object whose underlying tree-view control has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newTreeView* method.

notifyCode

The numeric notification code of the event that caused the notification to be sent.

Return:

The return from the EXPANDED event handler is ignored by the operating system, any value can be returned. 0 makes a good return value.

Remarks:

The event handlers for both the EXPANDED and the *EXPANDING* events are similar and are sent the same arguments. It is possible to connect both events to the same event handler and use the 6th *notifyCode* to determine which event caused the event handler to be invoked.

Example

The following example displays the expanded or collapsed action that has just happened:

```
::method onExpanded unguarded
  use arg id, item, what, extra, treeView, notifyCode
  say "The item with handle" item "has been" what

  return 0

/*
  Possible output:

  The item with handle 0x00000000001AD3E0 has been COLLAPSED
*/
```

4.7.33.7. Expanding Event Handler

The event handler for the EXPANDING event is invoked *before* a tree-view item is going to be expanded or collapsed.

The *willReply* argument in the *connectTreeViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onExpanding unguarded
  use arg id, hItem, what, extra, treeView, notifyCode

  return trueOrFalse
```

Arguments:

The event handling method receives 6 arguments:

id

The resource ID of the tree-view control sending the notification.

hItem

The handle of the tree-view item that is about to expand or collapse.

what

A string that indicates whether the item is going to expand or collapsed. The string will be either *EXPANDED* or *COLLAPSED*.

extra

The *extra* arugment will usually be the empty string. The Microsoft documentation seems to indicate that when an item is going to expand, the action may be expanded partial and when the action is collapsed, it may be collapsed and reset. If expanded partial is detected, the *extra* argument will be *PARTIAL*. If collapse and reset is detected, the *extra* argument will be *RESET*. However, in testing, neither of these 2 actions were ever detected and the Microsoft documentation is unclear here. It may be that the *extra* argument will always be the empty string.

treeView
: The Rexx tree-view object whose underlying tree-view control has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newTreeView* method.

notifyCode
: The numeric notification code of the event that caused the notification to be sent.

Return:

If the programmer used the *willReply* argument with a true value, the event handler must return true or false. This allows the application to control the expansion or collapse of the tree-view items.

Returning true, allows the expansion or collapse to happen. Returning false prevents the expansion or collapse. If the *willReply* argument was false, SYNC or omitted, the application can not control the action. Expansion or collapse is always allowed. The event handler can return any value, or not return a value at all. Good practice would be to always return a value from an event handler.

Remarks:

The event handlers for both the EXPANDING and the *EXPANDED* events are similar and are sent the same arguments. It is possible to connect both events to the same event handler and use the 6th *notifyCode* to determine which event caused the event handler to be invoked.

Example

The following example allows the expansion or collapse to continue and simply prints out what is about to happen:

```
::method onExpanding unguarded
  use arg id, item, what, extra, rxTv, notifyCode
  say "Item with handle" item "is going to be" what

  return .true

  /*
    Possible output:

    Item with handle 0x00000000001AD680 is going to be EXPANDED
  */
```

4.7.33.8. GetInfoTip Event Handler

The event handler method connected to the get info tip event is invoked when the tree-view control requests the text to display in the info tip. The programmer must return a string value and the interpreter waits for this return. The *willRepy* argument of the *connectTreeViewEvent* method is ignored for this event. The event handler must always return a value.

```
::method onGetInfoTip unguarded
  use arg id, hItem, text, maxLen, itemData, rxTv
  return infoText
```

Arguments:

The event handling method receives 6 arguments:

id
: The resource ID of the tree-view control requesting the info tip text.

hItem
: The handle of the tree-view item that the info tip is for.

text
: The current text the tree-view intends to display. Note that most often this is the empty string. However, in some cases it might not be the empty string. Microsoft suggests that if *text* is not the empty string, the application should append its text to the end of the string.

maxLen
: The maximum length of the string that will be displayed. The programmer should not assume what this length is. However, testing shows that it is usually 1023. If the returned text is longer than the *maxLen* value, the text will automatically be truncated to *maxLen* characters.

itemData
: The item *data* associated with the tree-view item that the info tip is for. If no item data has been associated with the item, then this argument will be the **.nil** object.

rxTv
: The Rexx tree-view object whose underlying tree-view control has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newTreeView* method.

Return:

The event handler must return a string value. If the returned string is not the empty string, it will be displayed as the info tip. If the empty string is returned, then the previous value of *text* is displayed. That is, if *text* is the empty string, no info tip will be shown. However, if *text* is not the empty string, that text will be displayed unchanged.

Example

In the following example, the item data associated with the tree-view item that the info tip is for, is inspected to see if it is the string: '...' If it is, then a info tip is displayed, otherwise no tip is displayed.

```
::method onGetInfoTip unguarded
    expose tv
    use arg id, hItem, text, maxLen, itemData

    if itemData == '...' then return 'There are too many books to list'
    else return ''
```

4.7.33.9. KeyDown Event Handler

The event handler for the key down event is invoked when the user types a key when the tree-view control has the keyboard focus. The event handler is similar to the event handler for the *KEYDOWNEX* event, it is invoked for the same event. However, when the KEYDOWN keyword is use, the event handler receives fewer arguments than when the KEYDOWNEX key word is used.

The *willReply* argument in the *connectTreeViewEvent* method determines how the event handler needs to respond to the notification.

```
::method onKeyDown unguarded
  use arg id, vKey, rxTv

  return 0
```

Arguments:

The event handling method receives 3 arguments:

id
: The resource ID of the tree-view control sending the notification.

vKey
: The virtual key code of the key typed.

rxTv
: The Rexx tree-view object whose underlying tree-view control has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newTreeView* method.

Return:

Using the KEYDOWN event keyword, as opposed to using the KEYDOWNEX keyword, causes the ooDialog framework to use the original ooDialog implementation. In the original implementation, the return from an event handler was ignored. Returning a value from the KEYDOWN event handler has no effect. So no return is required. However, good practice would be to always return a value from an event handler. See the *KEYDOWNEX* event handler for a discussion of what the operating system expects from a key down event handler.

Remarks:

The *VK* class can be used to determine which virtual key was pressed.

Example

The following example simply displays on the screen the key typed by the user:

```
::method onKeyDown unguarded
  use arg id, vkey
  say "Key" .VK~name2key(vkey) "was pressed."

  return 0
```

4.7.33.10. KeyDown (extended) Event Handler

The event-handling method connected through the KEYDOWNEX keyword is similar to the event handler for the KEYDOWN event handler. It is invoked for the same event, when the user presses a key within the tree-view. However, it receives a different set of arguments than that provided when the KEYDOWN keyword is used.

```
::method onKeyDownEx unguarded
  use arg vKey, isShift, isCtrl, isAlt, extraInfo, treeViewObj

  return response
```

Event Handler Method Arguments:

The event handling method receives 6 arguments:

vKey

The virtual key code of of the key pressed. The *VK* class can be used to determine which key was pressed.

isShift

A boolean (true or false) that denotes whether a shift key was down or up at the time of the key press. It will be true if a shift key was down and false if the shift key was not down.

isCtrl

True if a control key was down at the time of the key press, false if it was not.

isAlt

True if an alt key was down at the time of the key press, false if it was not.

extraInfo

This argument is a string containing keywords. It supplies extra information about the keyboard state at the time of a key press event. The string may be the empty string, otherwise it will contain some combination of these keywords:

extended

The key down event is for one of the extended keys.

numOn

Num Lock was on at the time of the key press event.

numOff

Num Lock was off.

capsOn

Caps Lock was on at the time of the key press event.

capsOff

Caps Lock was off.

scrollOn

Scroll Lock was on at the time of the key press event.

scrollOff

Scroll Lock was off.

lShift

The left shift key was down at the time of the key press event.

rShift

The right shift key was down.

lControl

The left control key was down at the time of the key press event.

rControl

The right control key was down.

lAlt

The left alt key was down at the time of the key press event.

rAlt
: The right alt key was down.

treeViewObj
: The Rexx **`TreeView`** object whose underlying Windows tree-view had the key down event.

Return:

When the *willReply* argument to the *connectTreeViewEvent* method is true, the event handler must return a value.

If the key pressed was a character key, the character is used as part of an incremental search. The event handler returns true to allow the character to be added to the incremental search and false to prevent the character from being added. If the key pressed was not a character key, then the return is ignored by the operating system. 0 makes a good return when the operating system ignores the return value.

If *willReply* was omitted, false, or SYNC, the event handler does not *have* to return a value, but good practice would be to always return a value from an event handler.

Example

The following example allows the user to type Ctrl-Ins (control insert) to add a new item to the tree view:

```
::method onKeyDownEx unguarded
    use arg vKey, isShift, isCtrl, isAlt, extraInfo, treeViewObj

    if vKey == .VK~INSERT, isCtrl then do
        reply .false

        self~addNewItem(treeViewObj)
        return
    end

    return .true
```

4.7.33.11. SelChanged Event Handler

The event handler for the selection changed event is invoked when the selection has changed from one item to another.

This event notification has been enhanced since the original ooDialog implementation. If the *willReply* argument to the *connectTreeViewEvent* method is omitted the old implementation is used. If the argument is not omitted, the new implementation is used. How the two event handlers work is described separately below:

New event handler description:

Whether the programmer must return a value and if the interpreter waits, or does not wait, for this return is determined by the value of the *willReply* argument. If *willReply* is true, the programmer must return a value from the event handler and the interpreter waits for that reply. If *willReply* is false the interpreter does not wait for a reply. If *willReply* was SYNC, the interpreter waits for the reply, but the actual value of the reply is disregarded.

```

    ::method onSelChanged unguarded
      use arg id, hItemOld, userOld, hItemNew, userNew, action, rxTv, nCode
    return 0
```

Arguments:

The event handling method receives 8 arguments:

id

The resource id of the tree-view sending the notification.

hItemOld

The handle of the tree-view item handle that has lost the selction. This may be 0 if no item had the selection.

userOld

The user *data* object assigned to the tree-view item that lost the selection. If no item data has been assigned to that item, of if *hItemOld* is 0, this argument will be the **`.nil`** object.

hItemNew

The handle of the tree-view item handle that gained the selction.

userNew

The user *data* object assigned to the tree-view item that gained the selection. If no item data has been assigned to that item, this argument will be the **`.nil`** object.

action

A keyword that indicates the type of action that caused the selection to change. This will be one of the following keywords:

KEYBOARD	MOUSECLICK	UNKNOWN

KEYBOARD

By a keystroke.

MOUSECLICK

By a mouseclick.

UNKNOWN

Unknown.

rxTv

The Rexx tree-view object whose underlying tree-view control has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newTreeView* method.

nCode

The numeric notification code of the event that caused the notification to be sent. Each dialog control has its own specific notification codes.

Return:

The operarting system ignores the returned value for this event notification. In these cases, 0 makes a good return. Good practice would be to always return a value from an event handler.

Old event handler description:

The old style event notification is used when the programmer omits the *willReply* argument in the invocation of the *connectTreeViewEvent* method. The return from the event handler is completely ignored, the interpreter does not wait for this return.

```
::method onSelChanged unguarded
  use arg id, useLess, rxTv, nCode
```

Arguments:

The event handling method receives 4 arguments:

id
: The resource id of the tree-view sending the notification.

useLess
: This argument is a number that has no meaning within the Rexx code. It is the value the old implementation of the event notification sent to the event handler.

rxTv
: The Rexx **`TreeView`** object whose underlying Windows tree-view sent the selection changed notification.

nCode
: The numeric notification code of the event that caused the notification to be sent. Each dialog control has its own specific notification codes.

Return:

The operating system ignores the returned value from this notification. Good practice would be to always return a value from any event handler. 0 is a good value for cases where the operating system ignores the returned value.

Remarks:

The event handlers for both the SELCHANGED and the *SELCHANGING* events are similar and are sent the same arguments. It is possible to connect both events to the same event handler and use the *nCode* argument to determine which event caused the event handler to be invoked.

4.7.33.12. SelChanging Event Handler

The event handler for the CHANGING event is invoked when the selection is about to change from one item to another. The notification is sent before the change is made.

This event notification has been enhanced since the original ooDialog implementation. If the *willReply* argument to the *connectTreeViewEvent* method is omitted the old implementation is used. If the argument is not omitted, the new implementation is used. Replying to the notification gives the programmer the opportunity to cancel the change in selection. How the two event handlers work is described separately below:

New event handler description:

The *willReply* argument in the *connectTreeViewEvent* method determines how the event handler needs to respond to the notification. Replying to this notification gives the application the opportunity to cancel the selection change

```
::method onSelChanging unguarded
  use arg id, hItemOld, userOld, hItemNew, userNew, action, rxTv, nCode

  return trueOrFalse
```

Arguments:

The event handling method receives 8 arguments:

id
: The resource id of the tree-view sending the notification.

hItemOld
: The handle of the tree-view item handle that is about to lose the selction. This may be 0 if no item has the selection.

userOld
: The user *data* object assigned to the tree-view item that is about to lose the selection. If no item data has been assigned to that item, of if *hItemOld* is 0, this argument will be the **`.nil`** object.

hItemNew
: The handle of the tree-view item handle that will gain the selction.

userNew
: The user *data* object assigned to the tree-view item that will gain the selection. If no item data has been assigned to that item, this argument will be the **`.nil`** object.

action
: A keyword that indicates the type of action that will cause the selection to change. This will be one of the following keywords:

KEYBOARD	MOUSECLICK	UNKNOWN

KEYBOARD
: By a keystroke.

MOUSECLICK
: By a mouseclick.

UNKNOWN
: Unknown.

rxTv
: The Rexx tree-view object whose underlying tree-view control has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newTreeView* method.

nCode
: The numeric notification code of the event that caused the notification to be sent. Each dialog control has its own specific notification codes.

Return:

When the programmer used true for the *willReply* argument, the event handler must return true or false. Return true to allow the selection change. Return false to cancel the selection change.

Otherwise, the interpreter does not pass the returned value on to the operating system, and the event handler is not required to return a value. However, good practice would probably be to always return a value from an event handler.

Old event handler description:

The old style event notification is used when the programmer omits the *willReply* argument in the invocation of the *connectTreeViewEvent* method. The return from the event handler is completely ignored, the interpreter does not wait for this return.

```
::method onSelChanging unguarded
  use arg id lParam, rxTv, nCode
```

Arguments:

The event handling method receives 4 arguments:

id

The resource id of the tree-view sending the notification.

lParam

This argument is a number that has no meaning within the Rexx code. It is the value the old implementation of the event notification sent to the event handler.

rxTv

The Rexx tree-view object whose underlying tree-view control has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newTreeView* method.

nCode

The numeric notification code of the event that caused the notification to be sent. Each dialog control has its own specific notification codes.

Return:

The interpreter does not wait for the return from the event handler, so the return has no meaning. Good practice would be to return a value anyway.

Remarks:

The event handlers for both the SELCHANGING and the *SELCHANGED* events are similar and are sent the same arguments. It is possible to connect both events to the same event handler and use the *nCode* argument to determine which event caused the event handler to be invoked.

4.7.34. connectUpDownEvent

```
>>--connectUpDownEvent(--id--,--event--+---------------+--+-------------+--)---><
                                       +-,-methodName--+  +-,-willReply-+
```

The *connectUpDownEvent* method connects an *event* notification message from an *UpDown* control to a method in the Rexx dialog.

Arguments:

The arguments are:

id [required]

The resource ID of the up-down control whose notification message is to be connected to a Rexx dialog's method. May be numeric or *symbolic*.

event [required]

Exactly one of the following keywords. The keyword specifies the event to be connected and case is not significant. Unlike most controls, the up-down control only has one event notification.

DELTAPOS

Sent when the position of the control is about to change. This happens when the user requests a change in the value by pressing the control's up or down arrow. The event handler must *return* a reply for this event. The interpreter waits for that reply.

The DELTAPOS notification is sent before the scroll message which actually changes the control's position. This allows the programmer to examine, allow, modify, or disallow the change in position.

methodName [optional]

The name of the method that is to be invoked whenever the specified notification is received from the up-down control. The programmer defines this method. If this argument is omitted, a method name is automatically generated that consists of the event keyword preceded by **on**. For instance, **`onDeltaPos`**. The method name can not be the empty string.

willReply [optional]

The *willReply* argument controls how the interpreter invokes the event handler for the connected event. However, currently the up-down control only has one event notification and the operating system expects a return value from that notification. When the event is connected the *willReply* argument is ignored. The event handler for DELTAPOS must always return a delta postion buffer.

Return value:

Returns **`.true`** if the event was connected correctly, otherwise **`.false`**.

Remarks:

See the sections on *connecting* and *coding* event handlers for additional information on event handlers.

Details:

Syntax errors are raised when incorrect usage is detected, including the use of an invalid symbolic ID or an unrecognized event keyword.

If the programmer does not provide a matching method in the Rexx dialog, a syntax condition will be raised if the connected event happens.

The underlying dialog receives the UDN_* messages as the notifications for the up-down events.

4.7.34.1. DeltaPos Event Handler

The event handler for the up-down DELTAPOS event is invoked when when the position of the control is about to change. The arguments the event handler receives allow the programmer to examine the proposed change in position, to modify the change, or to cancel the change all together.

The programmer must return a value from the event handler and the interpreter waits for this return. The *deltaPosReply* class method of the *UpDown* class is used to properly construct the return value from the event handler. This return value can not be manually constructed in Rexx code, so the programmer must use the *deltaPosReply* method to return a value from the event handler.

```
::method onDeltaPos unguarded
  use arg pos, delta, id, hwnd, rxUpd

  return buffer
```

Arguments:

The event handling method receives 5arguments:

pos

A signed whole number that contains the up-down control's current position.

delta

A signed whole number that contains the proposed change in the up-down control's position. This is positive if the user has clicked the up button or used the up arrow key. If the user has clicked the down button or used the down arrow key, this number will be negative.

id

The resource ID of the up-down control whose position is about to change.

hwnd

The window *handle* of the up-down control whose position is about to change.

rxUpd

The Rexx up-down object whose underlying up-down has sent the notification. This is a convenience for the programmer. It is the same Rexx object the programmer would receive through the *newUpDown* method.

Return:

A delta position buffer must be returned by the event handler. This buffer can only be constructed properly by using the *UpDown* class's *deltaPosReply* method. The arguments to *deltaPosReply* allow the programmer to return a value that makes no change to the new position, cancels altogether the change in position, or modifies the resulting new position.

Example

The following example examines the change in position in the up-down control and modifies it so that the position in the up-down control is always an even number. Note that the *deltaPosReply* method ignores the second and third arguments when the first argument is **.false**. So, in the code below, if *change* remains **.false**, then the values of *cancel* and *delta* do not matter.

```
::method initDialog
...
  self~connectUpDownEvent(IDC_UPD, "DELTAPOS", onPosChange)
...

::method onPosChange unguarded
  use arg pos, delta, id, hwnd

  change = .false
  cancel = .false

  if ((pos + delta) // 2) <> 0 then do
    change = .true
    if delta > 0 then delta += 1
    else delta -= 1
  end

  return .UpDown~deltaPosReply(change, cancel, delta)
```

4.7.35. defListDragHandler

```
>>-defListDragHandler(--id--,--item--,--point--,--isLMB--,--listView--)--------><
```

A list-view control does not handle a drag-and-drop operation itself. It defers that handling to the programmer by sending a BEGINGDRAG, (left mouse drag,) or a BEGINRDRAG, (right mouse drag,) event notification. The programmer can code his own *event* handling method and use *connectListViewEvent*() method to connect the method to the drag-and-drop operation.

The *defListDragHandler* method is an event handling method supplied by the ooDialog framework and the programmer can use this method rather than code his own.

This method implementation allows the user to drag an item from one location to another within an icon view and a small icon view. The cursor shape is changed to a crosshair during the drag operation. The user can cancel the drag operation by clicking the other mouse button while holding the button that started the drag operation. Note that the final icon position is not flexible when the list-view control has the AUTOARRANGE style.

If the programmer wants to implement her own drag-and-drop event handler, she may want to examine the how the *defListDragHandler* method in the **`ooDialog.cls`** file. In addition the **`oodListViews.rex`** example program uses the *defListDragHandler*. Running the example program will show the behavior of the current *defListDragHandler* implementation.

Example:

```
::method initDialog

  ...

  self~connectListViewEvent(ID_LV_ICON, "BEGINDRAG", "defListDragHandler")
  -- The ooDialog framework now automatically handles Drag and drop operations.
```

4.7.36. defTreeDragHandler

```
>>-defTreeDragHandler(--id--,--item--,--point--)----------------><
```

A tree view control does not handle a drag-and-drop operation itself. It defers that handling to the programmer by sending a BEGINGDRAG, (left mouse drag,) or a BEGINRDRAG, (right mouse drag,) event notification. The programmer can code his own *event* handling method and use *connectTreeViewEvent*() method to connect the method to the drag-and-drop operation.

The *defTreeDragHandler* method is an event handling method supplied by the ooDialog framework and the programmer can use this method rather than code his own.

This method implementation allows the user to move an item, or a node with all its subitems, from one parent node to another within the tree view. The cursor shape is changed to a crosshair during the drag operation. If the cursor is moved over the item dragged, the cursor shape is changed to a slashed circle. The user can cancel the drag operation by clicking the other mouse button while holding the button that started the drag operation.

If the programmer wants to implement her own drag-and-drop event handler, she may want to examine the how the *defTreeDragHandler* method in the **`ooDialog.cls`** file. In addition both the **`oodtree.rex`** and **`propdemo.rex`** example programs uses the *defTreeDragHandler*. Running these example programs will show the behavior of the current *defTreeDragHandler* implementation.

Example:

```
::method initDialog

  ...

  self~connectTreeViewEvent("IDC_TREE","BEGINDRAG", "defTreeDragHandler")
  -- Drag and drop operations are not automatically handled.
```

4.8. PlainBaseDialog Class

PlainBaseDialog is the base class of all dialog classes in the ooDialog framework. It implements methods that are common to every dialog. PlainBaseDialog is an abstract class. A programmer can not instantiate a new PlainBaseDialog object and display a dialog on the screen. Rather, the ooDialog programmer subclasses one of the concrete classes provided by the ooDialog framework, such as the *ResDialog*, the *RcDialog*, or the *UserDialog*, to create and execute a real dialog.

Many of the methods of the PlainBaseDialog are actually implemented in one of the mixin classes inherited by the PlainBaseDialog. See this *diagram*/> to visualize the inheritance. In essence the PlainBaseDialog is the *dialog* object. All the methods and attributes of the PlainBaseDialog are listed in the dialog object chapter. This includes the methods and attributes of its inherited mixin classes.

The PlainBaseDialog inherits the following mixin classes:

- *DialogExtensions*
- *EventNotification*
- *ResourceUtils*
- *WindowBase*
- *WindowsExtensions*

4.8.1. Method Table

The following table lists the constant methods, class methods, attribute methods, and instance methods of the **`PlainBaseDialog`** class:

Table 4.11. PlainBaseDialog Method Reference

Dialog Method	Description
Constant Methods	The **`PlainBaseDialog`** provides a number of *constant* values through the **`::constant`** directive.
Class Methods	**Class Methods**
new	Instantiates a new dialog object.
getFontName	Returns the default font for all dialogs.
getFontSize	Returns the default font size for all dialogs.
setDefaultFont	Sets the default font name and size for all dialogs.
Attribute Methods	**Attribute Methods**
autoDetect	If automatic data field detection is on or off.
dlgHandle	The window handle of the dialog.
fontName	Name of the dialog's font.
fontSize	Point size of the dialog's font.

Dialog Method	Description
ownerDialog	The owner dialog of the dialog, if there is one.
Instance Methods	**Instance Methods**
addComboEntry	Adds a string to the list of a combo box.
addListEntry	Adds a string to the specified list box.
addNewAttribute	Adds a new attribute (a method) to this dialog.
addNewMethod	Adds a new method to this dialog.
autoDetection	Turns automatic data field detection on
backgroundBitmap	Sets a bitmap as the dialog's background picture.
backgroundColor	Sets the background color of a dialog.
cancel	A default event handler, provided by the ooDialog framework , for the cancel event.
center	Moves the dialog to the screen center.
changeComboEntry	changeComboEntry
changeListEntry	changeListEntry
comboAddDirectory	Adds all or selected file names in the given directory to the combo box.
comboDrop	Deletes all items from the list of the given combo box.
connectCheckBox	Creates a data attribute in the Rexx dialog object and connects it to a check box control in the underlying dialog.
connectComboBox	Creates a data attribute in the Rexx dialog object and connects it to a combo box control in the underlying dialog.
connectDateTimePicker	Creates a data attribute in the Rexx dialog object and connects it to a date time picker control in the underlying dialog.
connectEdit	Creates a data attribute in the Rexx dialog object and connects it to a edit control in the underlying dialog.
connectListBox	Creates a data attribute in the Rexx dialog object and connects it to a list box control in the underlying dialog.
connectListView	Creates a data attribute in the Rexx dialog object and connects it to a list view control in the underlying dialog.
connectMonthCalendar	Creates a data attribute in the Rexx dialog object and connects it to a month calendar control in the underlying dialog.
connectRadioButton	Creates a data attribute in the Rexx dialog object and connects it to a radio button control in the underlying dialog.
connectTab	Creates a data attribute in the Rexx dialog object and connects it to a tab control in the underlying dialog.
connectTrackBar	Creates a data attribute in the Rexx dialog object and connects it to a track bar control in the underlying dialog.
connectTreeview	Creates a data attribute in the Rexx dialog object and connects it to a tree view control in the underlying dialog.
connectUpDown	Creates a data attribute in the Rexx dialog object and connects it to a up-down control in the underlying dialog.
deleteComboEntry	Deletes a string from the combo box.
deleteListEntry	Deletes an item from a list box.

Dialog Method	Description
disableControl	Disables the dialog control with the specified ID.
dlgUnit2pixel	Takes a dimension expressed in dialog units of this dialog and transforms it to a dimension expressed in pixels.
enableControl	Enables the control with the specified resource id.
ensureVisible	Causes the dialog to reposition itself so that it is entirely on the visible screen.
execute	Creates the underlying dialog, shows it, starts the automatic methods, if any, and destroys the underlying dialog when the user closes it.
findComboEntry	Returns the index corresponding to a given text string in the combo box.
findListEntry	Returns the index of the specified string within the list box with the specified ID.
focusControl	Sets the input focus to the specified dialog control.
get	Returns the window handle of the Windows dialog associated with the top Rexx dialog instance.
getCheckBoxData	Sets the value of the connected data attribute in the Rexx dialog object to match the state of the specified check box in the underlying dialog.
getComboBoxData	Sets the value of the connected data attribute in the Rexx dialog object to match the state of the specified combo box in the underlying dialog.
getComboEntry	Returns the string at the specified index of the combo box.
getComboItems	Returns the number of items in the combo box.
getControlData	Sets the value of the connected data attribute in the Rexx dialog object to match the state of the specified dialog control in the underlying dialog.
getControlHandle	Retrieves the window handle of the specified dialog control.
getControlID	Returns the numeric resource ID of the control with the specified window handle.
getControlText	Gets the text of the specified dialog control
getCurrentComboIndex	Returns the index of the currently selected item in the comb box.
getCurrentListIndex	Returns the index of the currently selected list box item, or 0 if no item is selected.
getData	Sets all the Rexx dialog data attribute values to the state of the underlying, connected, Windows dialog controls.
getDataAttribute	Sets a data attribute of the Rexx dialog using the data from a connected Windows dialog control.
getDataStem	Sets the values of the indexes of the specified stem to the state of the underlying, connected, Windows dialog controls.
getFocus	Returns the window handle of the dialog control that currently has the input focus.
getListBoxData	Sets the value of the connected data attribute in the Rexx dialog object to match the state of the specified list box in the underlying dialog.
getListEntry	Returns the string at the specified index of the list box.
getListItems	Returns the number of items in the list box.
getMenuBar	Returns the menu object attached to the dialog, or .nil if there is no menu attached.

Dialog Method	Description
getRadioButtonData	Sets the value of the connected data attribute in the Rexx dialog object to match the state of the specified radio button in the underlying dialog.
getSelf	Returns the window handle of the dialog.
getTextSizeDlg	Calculates the size needed to display a string in dialog units.
getTextSizeDu	Calculates the size needed to display a string in dialog units **(preferred method.)**
getWindowText	Gets the text of the specified window.
hasMenuBar	Tests if the dialog has a menu bar attached.
help	A default implementation of the help event handler method provided by the ooDialog framework.
HideControl	Hides the specified control by marking it invisible and immediately redrawing the area it occupies.
hideControlFast	Marks the specified control as invisible without any redrawing.
hideWindow	Hides a whole dialog window or a dialog control window and forces the window to repaint.
hideWindowFast	Hides a whole dialog window or a dialog control window, but does not force the window to redraw.
initAutoDetection	Switches automatic data field detection on or off.
initDialog	A method automatically invoked by the ooDialog framework when the underlying dialog is first created.
insertComboEntry	Inserts a string into the list of a combo box.
insertListEntry	Inserts a string into the specified list box.
isDialogActive	Returns true if the underlying Windows dialog still exists.
isMaximized	Checks if a dialog is currently maximized.
isMinimized	Checks if a dialog is currently minimized.
leaving	A method automatically invoked by the ooDialog framework when the underlying dialog is being closed.
listAddDirectory	Adds all or selected file names of a given directory to the list box.
listDrop	Removes all items from the list box.
maximize	Maximizes the dialog on the screen.
minimize	Minimizes the dialog to the taskbar.
newCheckBox	Returns an object of the **`CheckBox`** class for the check box control with the specified resource ID.
newComboBox	Returns an object of the **`ComboBox`** class for the combo box control with the specified resource ID.
newDateTimePicker	Returns an object of the **`DateTimePicker`** class for the date time picker control with the specified resource ID.
newEdit	Returns an object of the **`Edit`** class for the edit control with the specified resource ID.
newGroupBox	Returns an object of the **`GroupBox`** class for the group box control with the specified resource ID.

Dialog Method	Description
newListBox	Returns an object of the **`ListBox`** class for the list box control with the specified resource ID.
newListView	Returns an object of the **`ListView`** class for the list-view control with the specified resource ID.
newMonthCalendar	Returns an object of the **`MonthCalendar`** class for the month calendar control with the specified resource ID.
newProgressBar	Returns an object of the **`ProgressBar`** class for the progress bar control with the specified resource ID.
newPushButton	Returns an object of the **`Button`** class for the push button control with the specified resource ID.
newRadioButton	Returns an object of the **`RadioButton`** class for the radio button control with the specified resource ID.
newScrollBar	Returns an object of the **`ScrollBar`** class for the scroll bar control with the specified resource ID.
newTrackBar	Returns an object of the **`TrackBar`** class for the track bar control with the specified resource ID.
newStatic	Returns an object of the **`Static`** class for the static control with the specified resource ID.
newTab	Returns an object of the **`Tab`** class for the tab control with the specified resource ID.
newToolBar	Returns an object of the **`ToolBar`** class for the toolbar control with the specified resource ID.
newToolTip	Returns an object of the **`ToolTip`** class for the tooltip control with the specified resource ID.
newTreeView	Returns an object of the **`TreeView`** class for the tree-view control with the specified resource ID.
newUpDown	Returns an object of the **`UpDown`** class for the up-down control with the specified resource ID.
noAutoDetection	Turns automatic data field detection off
ok	A default event handler, provided by the ooDialog framework, for the cancel event.
pixel2dlgUnit	Takes a dimension expressed in pixels and transforms it to a dimension expressed in dialog units of this dialog.
restore	Restores a minimized or maximized dialog to its original position.
sendMessageToControl	Sends a Windows message to a dialog control and returns its response as a whole number.
sendMessageToControlH	Sends a Windows message to a dialog control and returns its response as a handle.
sendMessageToWindow	Sends a Windows message to a window and returns its response as a whole number.
sendMessageToWindowH	Sends a Windows message to a window and returns its response as a handle.
setCheckBoxData	Sets the *data* of a check box control.
setComboBoxData	Sets the *data* of a combo box control.

Dialog Method	Description
setControlData	Sets the *data* of a dialog control.
setControlText	Sets the text for the specified dialog control
setCurrentComboIndex	Selects the item with the specified index within the combo box list. If called without an index, all items in the combo box list are deselected.
setCurrentListIndex	Selects the item with the specified index in the list box. If called without an index, all items in the list box are deselected.
setData	Sets the state of all underlying, connected, dialog controls to the values of the matching dialog object data attributes.
setDataAttribute	Sets the state of a dialog control using the value of a data attribute of the Rexx dialog.
setDataStem	Sets the *data*, (the state,) of a number of Windows dialog controls to the values specified by a stem.
setDlgFont	Sets the font that to be used for the underlying Windows dialog, when it is created.
setEditData	Sets the *data* of an edit control.
setFocus	sets the input focus to a dialog control specified by hwnd and returns the window handle of the control that previously had the focus.
setFocusToWindow	Moves the input focus to another top-level window or dialog.
setGroup	Adds or removes the *group* style for the control specified.
setListBoxData	Sets the *data* of the underlying list box to the value specified.
setListTabulators	Sets the tabulators for a list box.
setRadioButtonData	Sets the *data* of a radio button control.
setTabStop	Add or remove the tab stop style for the specified control.
setWindowText	Sets the text for the specified window.
show	Sets the dialog window's show state.
showControl	Makes the specified dialog control reappear on the screen.
showControlFast	Shows a dialog control without redrawing its area.
showWindow	Shows the window or dialog control again.
showWindowFast	Marks the specified window as visible but does not force it to redraw.
tabToNext	Sets the focus to the next tab stop dialog control in the dialog and returns the window handle of the dialog control that currently has the focus.
tabToPrevious	Sets the focus to the previous tab stop dialog control in the dialog and returns the window handle of the dialog control that currently has the focus.
tiledBackgroundBitmap	Sets a bitmap as the dialog's background brush.
toTheTop	Makes the dialog the topmost dialog.
validate	A method meant to be over-ridden. Its purpose is to validate the user's input and decide whether or not to allow the dialog to close.

4.8.2. getFontName (Class method)

```
PlainBaseDialog::getFontName


>>--getFontName----------------------------------><
```

4.8.3. getFontSize (Class method)

```
PlainBaseDialog::getFontSize


>>--getFontSize----------------------------------><
```

4.8.4. new (Class method)

```
PlainBaseDialog::new


>>--new(--library--,--id--+------------+--+----------+--)---------------------><
                          +-,--dlgData.-+  +-,--hFile-+
```

4.8.5. setDefaultFont (Class method)

```
PlainBaseDialog::setDefaultFont


>>--setDefaultFont(--fontName--,--fontSize--)----><
```

4.8.6. autoDetect (Attribute)

```
PlainBaseDialog::autoDetect


>>--autoDetect-----------------------------------><

>>--autoDetect = onOff---------------------------><
```

4.8.7. dlgHandle (Attribute)

```
PlainBaseDialog::dlgHandle
```

4.8.8. fontName (Attribute)

```
PlainBaseDialog::fontName


>>--fontName-------------------------------------><

>>--fontName = nameVar---------------------------><
```

4.8.9. fontSize (Attribute)

```
PlainBaseDialog::fontSize


>>--fontSize-------------------------------------><

>>--fontSize = sizeVar---------------------------><
```

4.8.10. ownerDialog (Attribute)

```
PlainBaseDialog::ownerDialog


>>--ownerDialog----------------------------------><

>>--ownerDialog = rexxDlg------------------------><
```

4.8.11. addComboEntry

```
PlainBaseDialog::addComboEntry


>>--addComboEntry(--id--,--aString--)------------><
```

4.8.12. addListEntry

```
PlainBaseDialog::addListEntry


>>--addListEntry(--id--,--aString--)-------------><
```

4.8.13. addNewAttribute

```
PlainBaseDialog::addNewAttribute


>>--addNewAttribute(--atrName--+------------+--+--------------+--+------------+--)--><
                               +-,-initVal--+  +-,-unguarded--+  +-,-private--+
```

4.8.14. addNewMethod

```
PlainBaseDialog::addNewMethod


>>--addNewMethod(--mthName--+--------------+--+------------+--)----------------><
                            +-,-unguarded--+  +-,-private--+
```

4.8.15. backgroundBitmap

```
PlainBaseDialog::backgroundBitmap
```

```
>>--backgroundBitmap(--bmpFilename--+-------------+--)-------------------------><
                                    +-,--"USEPAL"-+
```

4.8.16. autoDetection

```
PlainBaseDialog::autoDetection


>>--autoDetection--------------------------------><
```

4.8.17. backgroundColor

```
PlainBaseDialog::backgroundColor


>>--backgroundColor(--color--)-------------------><
```

4.8.18. cancel

```
PlainBaseDialog::cancel


>>--cancel---------------------------------------><
```

4.8.19. center

```
PlainBaseDialog::center


>>--center(--+------------+--)-------------------><
             +--showOpts--+
```

4.8.20. changeComboEntry

```
PlainBaseDialog::changeComboEntry


>>--changeComboEntry(--id--,--+-------+--,--aString--)-------------------------><
                              +-index-+
```

4.8.21. changeListEntry

```
PlainBaseDialog::changeListEntry


>>--changeListEntry(--id--,--+-------+--,--aString--)--------------------------><
                             +-index-+
```

4.8.22. comboAddDirectory

```
PlainBaseDialog::comboAddDirectory


>>--comboAddDirectory(--id--,--drvpath--,--fileAtrs--)---------><
```

4.8.23. comboDrop

```
PlainBaseDialog::comboDrop


>>--comboDrop(--id--)----------------------------><
```

4.8.24. connectCheckBox

```
PlainBaseDialog::connectCheckBox


>>--connectCheckBox(--id--+-------------------+--)---------------><
                          +-,--attributeName--+
```

4.8.25. connectComboBox

```
PlainBaseDialog::connectComboBox


>>--connectComboBox(--id--+------------------+--)---------------><
                          +-,-attributeName--+
```

4.8.26. connectDateTimePicker

```
PlainBaseDialog::connectDateTimePicker


>>--connectDateTimePicker(--id--+------------------+--)---------><
                                +-,--attributeName-+
```

4.8.27. connectEdit

```
PlainBaseDialog::connectEdit


>>--connectEdit(--id--+------------------+--)----><
                      +-,--attributeName-+
```

4.8.28. connectListBox

```
PlainBaseDialog::connectListBox
```

```
>>--connectListBox(--id--+------------------+--)---------------><
                         +-,--attributeName--+
```

4.8.29. connectListView

```
PlainBaseDialog::connectListView


>>--connectListView(--id--+-----------------+--)---------------><
                          +-,--attributeName-+
```

4.8.30. connectMonthCalendar

```
PlainBaseDialog::connectMonthCalendar


>>--connectMonthCalendar(--id--+-----------------+--)----------><
                               +-,--attributeName-+
```

4.8.31. connectRadioButton

```
PlainBaseDialog::connectRadioButton


>>--connectRadioButton(--id--+------------------+--)-----------><
                             +-,--attributeName--+
```

4.8.32. connectTab

```
PlainBaseDialog::connectTab


>>--connectTab(--id--+-----------------+--)-----><
                     +-,--attributeName-+
```

4.8.33. connectTrackBar

```
PlainBaseDialog::connectTrackBar


>>--connectTrackBar(--id--+-----------------+--)---------------><
                          +-,--attributeName-+
```

4.8.34. connectTreeview

```
PlainBaseDialog::connectTreeView


>>--connectTreeView(--id--+-----------------+--)---------------><
                          +-,--attributeName-+
```

4.8.35. connectUpDown

```
PlainBaseDialog::connectUpDown


>>--connectUpDown(--id--+-----------------+--)--><
                        +-,--attributeName-+
```

4.8.36. deleteComboEntry

```
PlainBaseDialog::deleteComboEntry


>>--deleteComboEntry(--id--,--index--)----------><
```

4.8.37. deleteListEntry

```
PlainBaseDialog::deleteListEntry


>>--deleteListEntry(--id--,--index--)-----------><
```

4.8.38. disableControl

```
PlainBaseDialog::disableControl


>>--disableControl(--id--)----------------------><
```

4.8.39. dlgUnit2pixel

```
PlainBaseDialog::dlgUnit2pixel


>>--dlgUnit2pixel(--du--)-----------------------><
```

4.8.40. enableControl

```
PlainBaseDialog::enableControl


>>--enableControl(--id--)-----------------------><
```

4.8.41. ensureVisible

```
PlainBaseDialog::ensureVisible


>>--ensureVisible-------------------------------><
```

4.8.42. execute

```
PlainBaseDialog::execute


>>--execute(--+-------------+--+---------+--)----><
              +--showOption-+  +-,--icon-+
```

4.8.43. findComboEntry

```
PlainBaseDialog::findComboEntry

r
>>--findComboEntry(--id--,--aString--)-----------><
```

4.8.44. findListEntry

```
PlainBaseDialog::findListEntry


>>--findListEntry(--id--,--aString--)------------><
```

4.8.45. focusControl

```
PlainBaseDialog::focusControl


>>--focusControl(--id--)-------------------------><
```

4.8.46. get

```
PlainBaseDialog::get


>>--get------------------------------------------><
```

4.8.47. getCheckBoxData

```
PlainBaseDialog::getCheckBoxData


>>--getCheckBoxData(--id--)----------------------><
```

4.8.48. getComboBoxData

```
PlainBaseDialog::getComboBoxData


>>--getComboBoxData(--id--)----------------------><
```

4.8.49. getComboEntry

```
PlainBaseDialog::getComboEntry


>>--getComboEntry(--id--,--index--)--------------><
```

4.8.50. getComboItems

```
PlainBaseDialog::getComboItems


>>--getComboItems(--id--)------------------------><
```

4.8.51. getControlData

```
PlainBaseDialog::getControlData


>>--getControlData(--id--)-----------------------><
```

4.8.52. getControlHandle

```
PlainBaseDialog::getControlHandle


>>--getControlHandle(--id--+---------+--)--------><
                           +-,--hDlg-+
```

4.8.53. getControlID

```
PlainBaseDialog::getControlID


>>--getControlID(--hwnd--)-----------------------><
```

4.8.54. getControlText

```
PlainBaseDialog::getControlText


>>--getControlText(--id--)-----------------------><
```

4.8.55. getCurrentComboIndex

```
PlainBaseDialog::getCurrentComboIndex


>>--getCurrentComboIndex(--id--)-----------------><
```

4.8.56. getCurrentListIndex

```
PlainBaseDialog::getCurrentListIndex


>>--getCurrentListIndex(--id--)------------------><
```

4.8.57. getData

```
PlainBaseDialog::getData


>>--getData-------------------------------------><
```

4.8.58. getDataAttribute

```
PlainBaseDialog::getDataAttribute


>>--getDataAttribute(--attributeName--)----------><
```

4.8.59. getDataStem

```
PlainBaseDialog::getDataStem


>>--getDataStem(--dataStem.--)-------------------><
```

4.8.60. getEditData

```
PlainBaseDialog::getEditData


>>-aBaseDialog~getEditData(--id--)----------------------------><
```

4.8.61. getFocus

```
PlainBaseDialog::getFocus


>>--getFocus------------------------------------><
```

4.8.62. getListBoxData

```
PlainBaseDialog::getListBoxData


>>--getListBoxData(--id--)-----------------------><
```

4.8.63. getListEntry

```
PlainBaseDialog::getListEntry


>>--getListEntry(--id--,--index--)---------------><
```

4.8.64. getListItems

```
PlainBaseDialog::getListItems


>>--getListItems(--id--)-------------------------><
```

4.8.65. getMenuBar

```
PlainBaseDialog::getMenuBar


>>--getMenuBar-----------------------------------><
```

4.8.66. getRadioButtonData

```
PlainBaseDialog::getRadioButtonData


>>--getRadioButtonData(--id--)-------------------><
```

4.8.67. getSelf

```
PlainBaseDialog::getSelf


>>--getSelf--------------------------------------><
```

4.8.68. getTextSizeDlg

```
PlainBaseDialog::getTextSizeDlg


>>--getTextSizeDlg(--text--+-------------+--+-------------+--+---------+--)----><
                           +-,-fontname--+  +-,-fontSize--+  +-,-hwnd--+
```

4.8.69. getTextSizeDu

```
PlainBaseDialog::getTextSizeDu


>>--getTextSizeDu(--text--)----------------------><
```

4.8.70. getWindowText

```
PlainBaseDialog::getWindowText


>>--getWindowText(--hwnd--)----------------------><
```

4.8.71. hasMenuBar

```
PlainBaseDialog::hasMenuBar


>>--hasMenuBar-----------------------------------><
```

4.8.72. help

```
PlainBaseDialog::help


>>--help-----------------------------------------><
```

4.8.73. HideControl

```
PlainBaseDialog::hideControl


>>--hideControl(--id--)--------------------------><
```

4.8.74. hideControlFast

```
PlainBaseDialog::hideControlFast


>>--hideControlFast(--id--)----------------------><
```

4.8.75. hideWindow

```
PlainBaseDialog::hideWindow


>>--hideWindow(--hwnd--)-------------------------><
```

4.8.76. hideWindowFast

```
PlainBaseDialog::hideWindowFast


>>--hideWindowFast(--hwnd--)---------------------><
```

4.8.77. initAutoDetection

```
PlainBaseDialog::initAutoDetection


>>-initAutoDetection---------------------------------><
```

4.8.78. initDialog

```
PlainBaseDialog::initDialog


>>--initDialog-----------------------------------><
```

4.8.79. insertComboEntry

```
PlainBaseDialog::insertComboEntry


>>--insertComboEntry(--id--,--+-------+--,--string--)-----------><
                              +-index-+
```

4.8.80. insertListEntry

```
PlainBaseDialog::insertListEntry


>>--insertListEntry(--id--,--+-------+--,--aString--)-------------------------><
                             +-index-+
```

4.8.81. isDialogActive

```
PlainBaseDialog::isDialogActive


>>--isDialogActive-------------------------------><
```

4.8.82. isMaximized

```
PlainBaseDialog::isMaximized


>>--isMaximized----------------------------------><
```

4.8.83. isMinimized

```
PlainBaseDialog::isMinimized
```

```
>>--isMinimized----------------------------------><
```

4.8.84. leaving

```
PlainBaseDialog::leaving

>>--leaving--------------------------------------><
```

4.8.85. listAddDirectory

```
PlainBaseDialog::listAddDirectory

>>--listAddDirectory(--id--,--drvPath--,--fileAtrs--)-----------><
```

4.8.86. listDrop

```
PlainBaseDialog::listDrop

>>--listDrop(--id--)-----------------------------><
```

4.8.87. maximize

```
PlainBaseDialog::maximize

>>--maximize-------------------------------------><
```

4.8.88. minimize

```
PlainBaseDialog::minimize

>>--minimize-------------------------------------><
```

4.8.89. newCheckBox

```
PlainBaseDialog::newCheckBox

>>--newCheckBox(--id--)--------------------------><
```

4.8.90. newComboBox

```
PlainBaseDialog::newComboBox
```

```
>>--newComboBox(--id--)--------------------------><
```

4.8.91. newDateTimePicker

```
PlainBaseDialog::newDateTimePicker


>>--newDateTimePicker(--id--)--------------------><
```

4.8.92. newEdit

```
PlainBaseDialog::newEdit


>>--newEdit(--id--)------------------------------><
```

4.8.93. newGroupBox

```
PlainBaseDialog::newGroupBox


>>--newGroupBox(--id--)--------------------------><
```

4.8.94. newListBox

```
PlainBaseDialog::newListBox


>>--newListBox(--id--)---------------------------><
```

4.8.95. newListView

```
PlainBaseDialog::newListView


>>--newListView(--id--)--------------------------><
```

4.8.96. newMonthCalendar

```
PlainBaseDialog::newMonthCalendar


>>--newMonthCalendar(--id--)---------------------><
```

4.8.97. newProgressBar

```
PlainBaseDialog::newProgressBar
```

```
>>--newProgressBar(--id--)-----------------------><
```

4.8.98. newPushButton

```
PlainBaseDialog::newPushButton


>>--newPushButton(--id--)------------------------><
```

4.8.99. newRadioButton

```
PlainBaseDialog::newRadioButton


>>--newRadioButton(--id--)-----------------------><
```

4.8.100. newScrollBar

```
PlainBaseDialog::newScrollBar


>>--newScrollBar(--id--)-------------------------><
```

4.8.101. newStatic

```
PlainBaseDialog::newStatic


>>--newStatic(--id--)----------------------------><
```

4.8.102. newTab

```
PlainBaseDialog::newTab


>>--newTab(--id--)-------------------------------><
```

4.8.103. newToolBar

```
PlainBaseDialog::newToolBar


>>--newToolBar(--id--)-------------------------------><
```

4.8.104. newToolTip

```
PlainBaseDialog::newToolTip
```

```
>>--newToolTip(--id--)-------------------------------><
```

4.8.105. newTrackBar

```
PlainBaseDialog::newTrackBar


>>--newTrackBar(--id--)--------------------------><
```

4.8.106. newTreeView

```
PlainBaseDialog::newTreeView


>>--newTreeView(--id--)--------------------------><
```

4.8.107. newUpDown

```
PlainBaseDialog::newUpDown


>>--newUpDown(--id--)----------------------------><
```

4.8.108. noAutoDetection

```
PlainBaseDialog::noAutoDetection


>>--noAutoDetection------------------------------><
```

4.8.109. ok

```
PlainBaseDialog::ok


>>--ok-------------------------------------------><
```

4.8.110. pixel2dlgUnit

```
PlainBaseDialog::pixel2dlgUnit


>>--pixel2dlgUnit(--pixels--)--------------------><
```

4.8.111. restore

```
PlainBaseDialog::restore
```

```
>>--restore--------------------------------------><
```

4.8.112. sendMessageToControl

```
PlainBaseDialog::sendMessageToControl


>>--sendMessageToControl(--id--,--msg--,--wParam--,--lParam--)-----------------><
```

4.8.113. sendMessageToControlH

```
PlainBaseDialog::sendMessageToControlH


>>--sendMessageToControlH(--id--,--msg--,--wParam--,--lParam--)----------------><
```

4.8.114. sendMessageToWindow

```
PlainBaseDialog::sendMessageToWindow


>>--sendMessageToWindow(--hwnd--,--msg--,--wParam--,--lParam--)----------------><
```

4.8.115. sendMessageToWindowH

```
PlainBaseDialog::sendMessageToWindowHsendMessageToWindowH)


>>--sendMessageToWindowH(--hwnd,--msg--,--wParam--,--lParam--)-----------------><
```

4.8.116. setCheckBoxData

```
PlainBaseDialog::setCheckBoxData


>>--setCheckBoxData(--id--,--data--)-------------><
```

4.8.117. setComboBoxData

```
PlainBaseDialog::setComboBoxData


>>--setComboBoxData(--id--,--data--)-------------><
```

4.8.118. setControlData

```
PlainBaseDialog::setControlData
```

```
>>--setControlData(--id--,--dataValue--)---------><
```

4.8.119. setControlText

```
PlainBaseDialog::setControlText


>>--setControlText(--id--,--text--)--------------><
```

4.8.120. setCurrentComboIndex

```
PlainBaseDialog::setCurrentComboIndex


>>--setCurrentComboIndex(--id--+----------+--)---><
                               +-,--index-+
```

4.8.121. setCurrentListIndex

```
PlainBaseDialog::setCurrentListIndex


>>--setCurrentListIndex(--id--+----------+--)----><
                              +-,--index-+
```

4.8.122. setData

```
PlainBaseDialog::setData


>>--setData--------------------------------------><
```

4.8.123. setDataAttribute

```
PlainBaseDialog::setDataAttribute


>>--setDataAttribute(--attributeName--)----------><
```

4.8.124. setDataStem

```
PlainBaseDialog::setDataStem


>>--setDataStem(--dataStem.--)-------------------><
```

4.8.125. setDlgFont

```
PlainBaseDialog::setDlgFont
```

```
>>--setDlgFont(--fontName--+-----------+--)------><
                           +-,-fontSize+
```

4.8.126. setEditData

```
PlainBaseDialog::setEditData


>>--setEditData(--id--,--data--)-----------------><
```

4.8.127. setFocus

```
PlainBaseDialog::setFocus


>>--setFocus(--hwnd--)---------------------------><
```

4.8.128. setFocusToWindow

```
PlainBaseDialog::setFocusToWindow


>>--setFocusToWindow(--hwnd--)-------------------><
```

4.8.129. setGroup

```
PlainBaseDialog::setGroup


>>--setGroup(--id--+-----------+--)--------------><
                   +-wantStyle-+
```

4.8.130. setListBoxData

```
PlainBaseDialog::setListBoxData


>>--setListBoxData(--id--,--data--)--------------><
```

4.8.131. setListTabulators

```
PlainBaseDialog::setListTabulators


                               +-,---+
                               V     |
>>--setListTabulators(--id--,----tab-+--)--------><
```

4.8.132. setRadioButtonData

```
PlainBaseDialog::setRadioButtonData


>>--setRadioButtonData(--id--,--data--)---------><
```

4.8.133. setTabStop

```
PlainBaseDialog::setTabStop


>>--setTabStop(--id--,--+-----------+--)---------><
                        +-wantStyle-+
```

4.8.134. setWindowText

```
PlainBaseDialog::setWindowText


>>--setWindowText(--hwnd--,--text--)-------------><
```

4.8.135. show

```
PlainBaseDialog::show


>>--show(--+--------+--)-------------------------><
           +--opts--+
```

4.8.136. showControl

```
PlainBaseDialog::showControl


>>--showControl(--id--)--------------------------><
```

4.8.137. showControlFast

```
PlainBaseDialog::showControlFast


>>--showItemFast(--id--)-------------------------><
```

4.8.138. showWindow

```
PlainBaseDialog::showWindow


>>--showWindow(--hwnd--)-------------------------><
```

4.8.139. showWindowFast

```
PlainBaseDialog::showWindowFast


>>-showWindowFast(--hwnd--)----------------------><
```

4.8.140. tabToNext

```
PlainBaseDialog::tabToNext


>>--tabToNext------------------------------------><
```

4.8.141. tabToPrevious

```
PlainBaseDialog::tabToPrevious


>>--tabToPrevious--------------------------------><
```

4.8.142. tiledBackgroundBitmap

```
PlainBaseDialog::tiledBackgroundBitmap


>>--tiledBackgroundBitmap(--bmpFilename--)-------><
```

4.8.143. toTheTop

```
PlainBaseDialog::toTheTop


>>--toTheTop-------------------------------------><
```

4.8.144. validate

```
PlainBaseDialog::validate


>>--validate-------------------------------------><
```

4.9. ResourceUtils Mixin Class

The **ResourceUtils** class provides utility methods for dealing with *resource IDs* and parsing resource *script* files. The class is inherited by the *dialog* object and the *ApplicationManager* class. Most of the methods of the **ResourceUtils** class are used *internally* and not documented.

4.9.1. Method Table

The following table list the methods of the **`ResourceUtils`** class for use by the programmer:

Table 4.12. ResourceUtils Methods

Method	Description
Attributes	**Attributes**
constDir	A directory object that maps symbolic resource IDS to their numeric IDs.
Instance Methods	**Instance Methods**
parseIncludeFile	Reads a file and adds any symbol definitions found to the proper constant directory.
resolveNumericID	Uses the constant directory to resolve a numeric ID to its symbolic ID.
resolveSymbolicID	Uses the constant directory to resolve a symbolic ID to its numeric value.

4.9.2. constDir (Attribute)

```
>>--constDir-------------------------------------><

>>--constDir[symbol] = numericValue--------------><
```

The *constDir* attribute is assigned a **`Directory`** object whose indexes are *symbolic* resource IDs. It is part of the*mechanism* provided by the ooDialog framework to allow the use of symbolic IDs in ooDialog programs. Although the use of the *constDir* is mostly transparent to the programmer, it is important to understand how the mechanism works when using symbolic IDs in ooDialog programs

constDir get:

Getting the **`constDir`** is public. In practical use, it is more common to get the value of an index of the *constDir*:

```
  id = self~constDir[IDC_RB_MINIMUM]
```

constDir set:

Setting the *constDir* attribute is private and should not be done by the programmer. The ooDialog framework internally manages the attribute, setting it to a **`Directory`** object at the proper point in program execution.

Remarks:

Both *dialog* objects and the *ApplicationManager* class inherit the *ResourceUtils* mixin class. **Note** that the *constDir* attribute of the *.application* **is** the global *.constdir*.

Each index in the **`Directory`** object assigned to the *constDir* attribute is the symbolic name of a resource ID. The item at that index is the numeric value of the *resource ID*. Once a symbolic ID is added to the *constDir* directory as an index, that symbolic ID can be used in place of the numeric resource ID as the resource ID argument in methods that use a resource ID.

This is dependent, however, on the global **`.constDir`** usage *strategy* the programmer has chosen. If the programmer has elected to not use the global **`.constDir`** at all, then symbolic IDs can only be used in the methods of the dialog or dialog control classes. If the programmer has elected to use the **`.constdir`** then symbolic IDs can be used in any class the ooDialog framework provides.

How to add symbolic IDs is explained in the overview chapter, in the discussion of the *mechanism* used by ooDialog to support symbolic IDs.

The global *.constdir* was introduced in ooDialog 4.2.0 to overcome limitations inherent in using the *constDir* attribute of the dialog object. The global **`.constDir`** is more efficient and capable than the *constDir* attribute. The programmer should be sure to review the *discussion* in the overview to determine how to take the best advantage of the mechanism ooDialog is providing for the use of symbolic IDs.

Details:

The *constDir* attribute is inherited from the *ResourceUtils* class.

4.9.3. parseIncludeFile

```
>>--parseIncludeFile(--fileName--)---------------><
```

Reads a file and adds any symbol *definitions*s found to the *constDir* attribute of this **`ResourceUtils`** object.

Arguments:

The single argument is:

fileName [required]

The name of the file to read and parse.

Return value:

Returns true if no errors were detected, otherwise false.

Remarks:

The ooDialog framework provides a *mechanism* allowing programmers to use *symbolic* IDs in their program. The *parseIncludeFile* method adds symbols to a constant directory, which is part of that mechanism. Which constant directory the symbol is added to is determined by the global **`.constDir`** usage *strategy* the programmer has chosen.

Example:

In this example the programmer has several different files with symbol definitions in them. When the first dialog object is instantiated, one of the symbol files is passed to the *new* method and is parsed automatically by the ooDialog framework. The other symbol files are then read and added to the constant directory:

```
dlg = .CueDialog~new("cueEdit.rc", IDD_CUE_DLG, , 'cueEdit.h')

dlg~parseIncludeFile('productionDialogs.h')
dlg~parseIncludeFile('debugDialogs.h')
dlg~parseIncludeFile('databaseDialogs.h')
```

4.9.4. resolveNumericID

```
>>--resolveNumericID(--id--)---------------------><
```

Attempts to resolve a number with an index in the *constDir* attribute of this **`ResourceUtils`** object.

Arguments:

The single argument is:

id [required]

The number to resolve. If id is not a number this method fails immediately.

Return value:

Returns the index, the symbolic *symbolic* ID, for the number, if the number can be matched to an item in the constant directory. -1 is returned on failure.

Remarks:

The ooDialog framework provides a *mechanism* allowing programmers to use *symbolic* IDs in their program. This method is intended to look up a numeric resource ID and return its symbolic ID. The lookup is done in the **`.Directory`** object assigned to the *constDir* attribute of this resource utils object. If an item in the **`constDir`** matches the numeric *id* then that item's index is returned.

The assumption in the description of this method, is that the programmer has only put items in the constant directory that are numeric IDs and whose indexes are symbol IDs. However, the lookup and return work the same no matter what is in the constant directory.

Although the operating system allows duplicate numeric resource IDs, as long as the IDs are for different types of resources, programmers must use unique ID numbers if they intend to use this method. If the constant directory contains duplicate numeric items, the return is non-deterministic.

Exactly which constant directory the lookup is done in is determined by the global **`.constDir`** usage *strategy* the programmer has chosen.

4.9.5. resolveSymbolicID

```
>>--resolveSymbolicID(--id--)--------------------><
```

Resolves *id* to its numeric value using the *constDir* attribute of this **`ResourceUtils`** object.

Arguments:

The single argument is:

id [required]

The id to resolve.

Return value:

The numeric resource ID on success, -1 on failure.

Remarks:

The ooDialog framework provides a *mechanism* allowing programmers to use *symbolic* IDs in their program. The *resolveSymbolicID* is the foundation of this mechanism. The framework relies on it heavily, however it is not expected that the programmer would have much need for the method.

The method first checks if *id* is a whole number greater than 0. If so the number is simply returned. If *id* is a whole number, but less than 1, then the method fails. If *id* is not whole number, then the constant directory of this resource utils object is checked to see if *id* is one of its indexes. If it is not the method fails. If it is an index, then the index's item is checked to see if it is a whole number greater than 0. If not the method fails. If the item is a whole number greater than 0, then the item is returned.

Exactly which constant directory is checked is determined by the global **`.constDir`** usage *strategy* the programmer has chosen.

4.10. WindowBase Mixin Class

WindowBase is a mixin class with methods that are common to all windows. It is inherited by both the *dialog* and dialog *control* objects. Every dialog object and every dialog control object have the instance methods of the WindowBase class. In addition, the *Window* class inherits the WindowBase class.

4.10.1. Method Table

The following table lists the attribute methods, and instance methods of the **`WindowBase`** Mixin Class.

Table 4.13. WindowBase Method Reference

Method	Description
Attribute Methods	**Attribute Methods**
factorX	The horizontal size of one dialog unit in pixels. (Inaccurate.)
factorY	The vertical size of one dialog unit in pixels. (Inaccurate.)
hwnd	The *handle* of this window.
initCode	Reflects if the dialog object initialization was successful. Has no meaning if the object is a dialog control.
pixelCX	The width of this window in pixels.
pixelCY	The height of this window in pixels.
sizeX	The width of this window in dialog units. (Inaccurate.)
sizeY	The height of this window in dialog units. (Inaccurate.)
Instance Methods	**Instance Methods**
childWindowFromPoint	Determines which, if any, of the child windows belonging to this window contains the specified point.
clear	Clears the client area of this window by painting it with the background brush.
client2screen	Converts a point, or points, in client-area coordinates of this window to its screen coordinates.
clientRect	Returns a **`Rect`** object containng the dimensions of this window's client area in pixels.
clientToScreen	Converts client-area coordinates of this window to its screen coordinates.
disable	Disables this window.
display	Shows or hides this window.
draw	Redraws the entire client area of this window immediately.
enable	Enables this window.
foregroundWindow	Returns the handle of the window in the foreground.
getClientRect	Returns the dimensions of this window's client area.
getExStyleRaw	Retrieves the numeric value of this window's extended style flags.
getID	Retrieves the identification number of this window.
getPos	Returns the position of this window in dialog units **(not accurate.)**
getRealPos	Returns the position of this window in pixels as a **`Point`** object.
getRealSize	Returns the size of this window in pixels as a **`Size`** object.

Method	Description
getRect	Returns the dimensions of this window.
getSize	Returns the size of this window in dialog units **(not accurate.)**
getStyleRaw	Retrieves the numeric value of this window's style flags.
getText	Gets the text of this window.
getTextSizePx	Calculates the size needed, in pixels, for a string written in this window. **(Preferred method.)**
getTextSizeScreen	Gets the size, width and height, in pixels, needed to display a string in a specific font.
hide	Hides this window by marking it invisible and immediately redrawing the area it occupies.
hideFast	Marks this window as invisible, no redrawing is done.
isEnabled	Tests if this window is enabled.
isVisible	Tests if this window is visible.
mapWindowPoints	Converts, or maps, a set of points from the coordinate space relative to this window to a coordinate space relative to the specified window.
move	Moves this window to the position specified in dialog units **(not accurate.)**
moveTo	Moves this window to the position specified in pixels.
moveWindow	Changes the position, visibility, and Z order of this window.
redraw	Redraws the entire window and all its child windows immediately.
redrawClient	Redraws the entire client area of this window immediately.
resize	Resizes this window to the size specified in dialog units **(not accurate.)**
resizeTo	Resizes this window to the size specified in pixels.
screen2client	Converts a point or points in screen coordinates to the client-area coordinates of this window.
screenToClient	Converts screen coordinates to the client-area coordinates of this window.
sendMessage	Sends a Windows message to the underlying window of this object and returns its response as a whole number.
sendMessageHandle	Sends a Windows message to the underlying window of this object and returns its response as a handle.
setRect	Moves and / or resizes this window.
setStyleRaw	Sets the value of this window's style flags using the numeric value specified.
setText	Sets the text, the caption, of this window.
setTitle	Sets the text of this window.
setWindowPos	Changes the size, position, visibility, and Z order of this window.
showFast	Marks this window as visible.
sizeWindow	Changes the size, visibility, and Z order of this window.
title	Gets the text of this window.
title=	Sets the text of this window.
update	Invalidates the entire client area of this window.
windowRect	Returns a `Rect` object containng the dimensions of this window in pixels.

4.10.2. factorX (Attribute)

```
>>--factorX--------------------------------------><

>>--factorX=ratio--------------------------------><
```

factorX was intended to be the horizontal size of one dialog unit, in pixels. However, the value is not *accurate* with modern dialogs, except under rare circumstances.

To accurately convert between dialog units and pixels use the appropriate *dlgUnit2pixel*() or *pixel2dlgUnit*() method.

4.10.3. factorY (Attribute)

```
>>--factorY--------------------------------------><

>>--factorY=ratio--------------------------------><
```

factorY was intended to be the vertical size of one dialog unit, in pixels. However, the value is not *accurate* with modern dialogs except under rare circumstances.

To accurately convert between dialog units and pixels use the appropriate *dlgUnit2pixel*() or *pixel2dlgUnit*() method.

4.10.4. hwnd (Attribute)

```
>>--hwnd-----------------------------------------><
```

The window *handle* of the underlying window.

hwnd get:

Before the underlying dialog object is created, the *hwnd* attribute for a dialog is not valid. Since dialog control objects can not be instantiated before the underlying controls is created, their handles are always valid.

hwnd set:

The programmer can not set the *hwnd* attribute, it is set internally by the ooDialog framework.

Example:

```
::method inspectHandles private
  okButton = self~newPushButton(IDOK)
  cancelButton = self~newPushButton(IDCANCEL)

  say "Window handles:"
  say "  ok button:     " okButton~hwnd
  say "  cancel button:" cancelButton~hwnd

/* Output might be:

Window handles:
  ok button:     0x00180298
  cancel button: 0x001A037A
```

```
*/
```

4.10.5. initCode (Attribute)

```
>>--initCode-------------------------------------><

>>--initCode=code--------------------------------><
```

A whole number representing the success of initialization of a dialog object.

Details

For a dialog *control* the *initCode* does not have any meaning.

initCode get:

For a dialog control, the value is always 0.

Remarks:

For a dialog object, after the *init* method has executed the *initCode* attribute will be 0 if the dialog initialization detected no errors. The attribute will be non-zero if initialization failed or an error was detected. The programmer should always check the *initCode* attribute after instantiating a dialog object. If the attribute is not zero, then the dialog was not initialized correctly and its behavior is undefined.

After the underlying dialog is closed, the *initCode* attribute will be 1 if the dialog was terminated with Ok and 2 if terminated with Cancel.

Example:

This example instantiates a new dialog and checks the init code for error:

```
style = "CENTER CONNECTBUTTONS CONNECTRADIOS CONNECTCHECKS"

dlg = .SimpleButtons~new("buttons.rc", IDD_DIALOG1, , "resource.h", style)

if dlg~initCode = 0 then do
  dlg~execute("SHOWTOP")
  dlg~Deinstall
end
else do
  say "Problem creating the dialog.  Init code:" dlg~initCode
  return 99
end

return 0
```

4.10.6. pixelCX (Attribute)

```
>>--pixelCX--------------------------------------><
```

The value of the *pixelCX* is the width of the window in *pixel*s.

Details

The *pixelCX* value is the correct width of the window, as opposed to the *sizeX* attribute which is usually *inaccurate* because it is calculated using *factorX*.

pixelCX get:

For a dialog object, before the underlying dialog is created the value of this attribute is 0.

pixelCX set:

The programmer can not set the *pixelCX* attribute, it is set internally by the ooDialog framework.

Remarks:

To convert the pixel width of a window to dialog units, the *dlgUnit2pixel*() method can be used. Although, once the underlying dialog is created, it is usually more convenient to work with pixels.

Example:

This example displays the pixel size of the ok button and then converts that size back to the dialog unit size of the button. Note that the converted size matches the size the button was actually created with.

```
::method defineDialog

  self~createPushButton(IDOK, 142, 99, 50, 14, "DEFAULT", "Ok")
  self~createPushButton(IDCANCEL, 197, 99, 50, 14, , "Cancel")

::method ok unguarded

  okButton = self~newPushButton(IDOK)

  say 'The ok button was created' 50 'dialog units wide.'
  say 'The ok button was created' 14 'dialog units high.'
  say
  say 'The pixel width of the button is: ' okButton~pixelCX
  say 'The pixel height of the button is:' okButton~pixelCY
  say
  say 'Convert pixels to dialog units.'
  size = .Size~new(okButton~pixelCX, okButton~pixelCY)
  self~pixel2dlgUnit(size)
  say 'The dialog unit width of the button is: ' size~width
  say 'The dialog unit height of the button is:' size~height

  self~ok:super

/* Output might be:

The ok button was created 50 dialog units wide.
The ok button was created 14 dialog units high.

The pixel width of the button is:  75
The pixel height of the button is: 23

Convert pixels to dialog units.
The dialog unit width of the button is:  50
The dialog unit height of the button is: 14

*/
```

4.10.7. pixelCY (Attribute)

```
>>--pixelCY--------------------------------------><
```

The value of the *pixelCY* is the height of the window in *pixel*s.

Details

The *pixelCY* value is the correct height of the window, as opposed to the *sizeY* attribute which is usually *inaccurate* because it is calculated using *factorY*.

pixelCY get:

For a dialog object, before the underlying dialog is created the value of this attribute is 0.

pixelCY set:

The programmer can not set the *pixelCY* attribute, it is set internally by the ooDialog framework.

Remarks:

To convert the pixel width of a window to dialog units, the *dlgUnit2pixel*() method can be used. Although, once the underlying dialog is created, it is usually more convenient to work with pixels.

Example:

The pixelCX *example* also uses the *pixelCY* attribute.

4.10.8. sizeX (Attribute)

```
>>--sizeX----------------------------------------><

>>--sizeX=dialogUnits----------------------------><
```

The *sizeX* attribute was intended to be the width of the window and *sizeY* the height of the window in *dialog unit*s. However, in almost all cases, the value is *inaccurate*)

Details

For a dialog control, *sizeX* and *sizeY* are calculated using *factorX* and *factorY*, which inherently makes them incorrect in almost all cases. *UserDialog* dialogs and their subclasses will have the correct values. However, the *sizeX* and *sizeY* values in a *ResDialog* dialog are 0.

These attributes are set one time by the ooDialog framework and not changed by the framework again.

To obtain the correct width and height of a window in either pixels or dialog units, use the *pixelCX* and *pixelCY* attributes. These values can be correctly converted, as shown in this *example*, to dialog units if dialog units are desired.

Example:

This example displays the width and height of the ok button using *sizeX* and *sizeY*. Contrast the output with the *example* for *pixelCx*.

```
::method defineDialog

  self~createPushButton(IDOK, 142, 99, 50, 14, "DEFAULT", "Ok")
  self~createPushButton(IDCANCEL, 197, 99, 50, 14, , "Cancel")

::method ok unguarded

  okButton = self~newPushButton(IDOK)

  say 'The ok button was created' 50 'dialog units wide.'
  say 'The ok button was created' 14 'dialog units high.'
  say
  say 'The pixel width of the button is: ' okButton~pixelCX
  say 'The pixel height of the button is:' okButton~pixelCY
  say
  say 'Use sizeX and sizeY to display dialog units.'
  say 'The dialog unit width of the button is: ' okButton~sizeX
  say 'The dialog unit height of the button is:' okButton~sizeY

  self~ok:super

  /* Output might be:

The ok button was created 50 dialog units wide.
The ok button was created 14 dialog units high.

The pixel width of the button is:  75
The pixel height of the button is: 23

Use sizeX and sizeY to display dialog units.
The dialog unit width of the button is:  37
The dialog unit height of the button is: 11
  */
```

4.10.9. sizeY (Attribute)

```
>>--sizeY----------------------------------------><

>>--sizeY=dialogUnits----------------------------><
```

The *sizeY* attribute is the counterpart to the *sizeX* attribute. The attribute was meant to be the height of the window in dialog *dialog unit*s, but is not accurate. The documentation for *sizeX* has a complete discussion of both attributes.

4.10.10. childWindowFromPoint

```
>>--childWindowFromPoint(--point--+----------+--)------------------------------><
                                  +-,-flags--+
```

Determines which, if any, of the child windows belonging to this window contains the specified point.

Arguments:

The arguments are:

point [required]

A *Point* object specifying the point to test for. The point must be in *coordinates* area coordinates of the parent window.

flags [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. This argument specifies which child windows to skip. When omitted, the flags are set to SKIPINVISIBLE SKIPDISABLED SKIPTRANSPARENT, that is disabled, invisible, and transparent child windows are ignored in the search.

ALL	SKIPINVISIBLE
SKIPDISABLED	SKIPTRANSPARENT

ALL

Does not skip any child windows.

SKIPDISABLED
: Skips disabled child windows.

SKIPINVISIBLE
: Skips invisible child windows.

SKIPTRANSPARENT
: Skips transparent child windows.

Return value:

Returns the window *handle* of the first child window meeting the search criteria. If the point lies within the parent window, but not within a child window meeting the search criteria, then the handle of the parent window is returned. If the point lies outside of the parent window, then 0 is returned.

Remarks:

The search is restricted to immediate child windows. Grandchildren and deeper descendant windows are not searched.

Since this method is a method of the *WindowBase* class, it is a method of both the *dialog* object and the dialog *control* object. It is of limited use with a dialog control because most dialog controls do not have child windows. However, some do. For instance, a *ListView* view control in report view contains a child header window.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

Example:

This example tests if a list view control is under the point 20, 25 of the dialog:

```
pt = .Point~new(20, 25)

hwnd = self~childWindowFromPoint(pt)
if hwnd = self~newListView(IDC_LV_ADDRESSES)~hwnd then do
  say 'The point 20, 25 is contained by the Addresses list view.'
end
else do
say 'The point 20, 25 is not contained by the Addresses list view.'
end
```

4.10.11. clear

```
>>--clear----------------------------------------><
```

The Clear method draws the dialog or dialog control using the background brush.

Return value:

0
: Clearing was successful.

1
: Clearing failed.

4.10.12. client2screen

```
Form 1:

>>--client2screen(--point--)---------------------><

Form 2:

>>--client2screen(--rect--)----------------------><

Generic form:

>>--client2screen(--obj--)-----------------------><
```

Converts the *client area* coordinates of the specified point(s) in this window to its screen coordinates.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Arguments:

The single argument specifies the client-area coordinates to be converted. It must be either a **`.Point`** or a **`.Rect`** object. A **`.Rect`** can be used to specify two points. Point one is taken to be the **`left`** and **`top`** attributes. Point two is taken to be the **`right`** and **`bottom`** attributes.

obj [required in/out]

On entry, the point(s) to be converted. On return, the converted points.

Return value:

True on success, false on failure.

Remarks:

This method uses the client-area coordinates given in the *obj* argument to compute screen coordinates. It then replaces the client-area coordinates in the object with the screen coordinates. The new coordinates are relative to the upper-left corner of screen. Coordinates are expressed in pixels. Additional comments.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

4.10.13. clientRect

```
>>--clientRect(--+--------+--)-------------------><
                 +--hwnd--+
```

Retrieves the coordinates of a window's client area as a *Rect* object. The coordinates are in pixels.

Arguments:

The single optional argument is:

hwnd [optional]

By default, the coordinates are for this window. However, the optional *hwnd* argument can be used to specify getting the coordinates for some other window.

Return value:

The coordinates of the client area of the window as a **`.Rect`** object.

Remarks:

The *clientRect* method supplies an alternative to the *getClientRect*(hwnd) method, to allow returning a **`.Rect`** object, which is usually more convenient, rather than a string of blank separated values.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example uses *clientRect* to get the size and position of the dialog's client area and then sets the edit control so that it takes up the entire client area.

```
::method resizeEditControl
  expose editControl isHidden

  hwnd = self~getControlHandle(IDC_MULTILINE)
  rect = self~clientRect

  self~setWindowRect(hWnd, rect)

  if isHidden then do
    editControl~show
    isHidden = .false
  end
```

4.10.14. clientToScreen

```
>>--clientToScreen(--x--,--y--)------------------><
```

The *clientToScreen* method maps the specified *client area* coordinates of this window to screen coordinates.

Arguments:

The arguments are:

x [required]

The x position of the point in the client-area, expressed in pixels.

y [required]

The y position of the point int the client-area of the window, expressed in pixels.

Return value:

The screen coordinates of the specified point, separated by a blank.

Remarks:

This old ooDialog method returns the coordinates as a single string that needs to be parsed. This is often not that convenient. The *client2screen* method is often easier to use.

4.10.15. disable

```
>>--disable--------------------------------------><
```

The *disable*() method disables the window. When a window is disabled it no longer responds to the mouse of the keyboard. The operating system usually draws disabled windows in a way that makes it easy for the user to see that the window is disabled.

Details:

Using this method before the underlying dialog is created does nothing.

Arguments:

There are no arguments.

Return:

True if the window was previously disabled, returns false if the window was not previously disabled. Note that this is not succes or failure return.

Example:

```
if self~isVisible then self~disable
```

4.10.16. display

```
>>--display(--+-----------+--)------------------><
              +--optString-+
```

The *display* method displays (shows, makes visible, or invisible) the window as specified.

Argument:

The single, optional, *optString* argument must be omitted or exactly one of the following keyword strings, case is not significant:

NORMAL

Makes the window visible in its default position and window size. If the window is a dialog, this has the effect of restoring the dialog size and position if it is minimized or maximized and giving it the focus. A dialog control window is simply made visible, if it was hidden. This is the default if the argument is omitted. This is exactly equivalent to the *show* method.

DEFAULT

DEFAULT is an alias for NORMAL. The two keywords are functionally identical.

HIDE

Makes the window invisible. This is exactly equivalent to the *hide* method.

NORMAL FAST

Marks the window as visible, but does not force it to be redrawn. If the window is already marked as visible this will have no effect. If the window is marked as invisible, then the next time the operating system draws the window it will become visible. Note that this is subtly different than using just NORMAL with a dialog window. If the dialog is minimized or maximized, it is not restored. In addition, it is not given the focus. This is exactly equivalent to the *showFast* method.

DEFAULT FAST

Same as using the NORMAL FAST option string.

HIDE FAST
: Marks the window as invisible, but does not force it to be redrawn. If the window is already marked as invisible this will have no effect. If the window is marked as visible, then the next time the operating system draws the window it will become visible. Note that this is subtly different than using just NORMAL with a dialog window. If the dialog is minimized or maximized, it is not restored. In addition, it is not given the focus. This is exactly equivalent to the *showFast* method.

INACTIVE
: This is the same as using the NORMAL option string, except that if the window is a dialog window it is not given the focus. In other words, for a dialog, if some other top-level window has the focus, the focus remains with that window. For a dialog control window there is no difference between INACTIVE and NORMAL. **Note** that INACTIVE FAST is **not** valid.

Return value:

When FAST is not in the option string, returns true if the window was previously visible and false if the window was previously hidden.

When FAST is in the option string, returns 0 for success, 1 on error. An error is extremely unlikely.

Remarks:

The *underlying* dialog must exist when this method is invoked.

When using the HIDE FAST options string, there are a number of caveats that may not be readily apparent. See the **Remarks** section in the *hideFast* section for a discussion of them.

Details:

Raises syntax errors for incorrect usage.

Sets the *.systemErrorCode* back to zero. However, the operating system does not set the error code for any of the APIs used by this method, so the variable will always be zero after invoking this method.

Example:

The following statement makes a tree view control in the dialog visible without redrawing it.

```
self~newTreeView(IDC_TREECONTROL_FILES)~display("NORMAL FAST")
```

This is usually done when there are a number of controls to make visible at one time. The programmer would make them all visible without redrawing. Then use the *update* method to schedule repainting everything at once.

4.10.17. draw

```
>>--draw-----------------------------------------><
```

The draw method causes the entire *client area* of the window to be immediately repainted. It is exactly the same as the *redrawClient* method, except that the background is never erased.

Details:

Using this method before the underlying dialog is created does nothing.

Sets the *.systemErrorCode*.

Arguments:

This method takes no arguments.

Return value:

The return values are:

0

Redrawing was successful.

1

Redrawing failed. Check **.systemErrorCode**.

Remarks:

Having a window redraw its client area has no effect if the window itself is hidden, there is nothing to draw. On the other hand, if a window, such as a dialog window, contains hidden windows that are still showing, then having it redraw has the effect of erasing the hidden windows, *provided* that the background is first erased.

4.10.18. enable

```
>>--enable---------------------------------------><
```

The *enable*() method enables the window. When a window is enabled it receives all user input, such as mouse clicks and key presses.

Details:

Using this method before the underlying dialog is created does nothing.

Arguments:

There are no arguments.

Return:

True if the window was previously disabled, returns false if the window was not previously disabled. Note that this is not succes or failure return.

Example:

```
lastName = self~newEdit(IDC_EDIT_LASTNAME)~getText

if self~isValidLastName(lastName) then self~newEdit(IDC_EDIT_FIRSTNAME)~enable
```

4.10.19. foregroundWindow

```
>>--foregroundWindow-----------------------------><
```

Returns the window *handle* of the current foreground window.

Arguments:

This method takes no arguments.

Return value:

The handle of the foreground window, or 0 if this method failed.

4.10.20. getClientRect

```
>>--getClientRect(--+------+--)------------------><
                    +-hwnd-+
```

The getClientRect method returns the client rectangle of a dialog or dialog control in screen pixels. The client coordinates specify the upper left and lower right corners of the client area. Because the client coordinates are relative to the upper left corner of the client area of a dialog or dialog control, the coordinates of the upper left corner are (0,0).

Arguments:

The only argument is:

hwnd

The handle to a dialog or dialog control. If this argument is omitted, the dimensions of the associated dialog or dialog control are returned.

Return value:

The client rectangle in the format "*left top right bottom*", separated by blanks.

4.10.21. getExStyleRaw

```
>>--getExStyleRaw--------------------------------><
```

The *getExStyleRaw* method retrieves the extended window style flags of the window. The flags are returned in their numeric format. This function is very similar to the *getStyleRaw*() method, allowing the programmer to get the extended window flags. The numeric value of the extended style flags can be looked up using the Windows *documentation* and the Platform *SDK*.

Details:

Using this method before the underlying dialog is created returns 0.

Arguments:

There are no arguments.

Return value:

The numeric value of the window extended style flags.

Example:

The following example gets the extended style flags for a list-view control and displays them to the screen in hexadecimal format. The value displayed in the example indicates the following styles.

WS_EX_NOPARENTNOTIFY == 0x00000004
WS_EX_CLIENTEDGE == 0x00000200

```
::method printExStyle private
  expose list

  val = list~getExStyleRaw

  numeric digits 11
  val = '0x' || val~d2x~right(8, 0)
  say "List control extended style flag:"
  say " " val
  say

/* Output could be: */

List control extended style flags:
  0x00000204
```

4.10.22. getID

```
>>--getID----------------------------------------><
```

Retrieves the identification number of the window. For a dialog window this will almost certainly be zero. For a dialog control window it will be the *resource ID* of the control.

Details:

Using this method before the underlying dialog is created returns 0.

Arguments:

There are no arguments.

Return value:

The numeric ID of the window.

Example:

The following example shows the IDs of the Ok button and the dialog itself. In most cases it will display 1 on the first line and 0 on the second line:

```
::method exampleInfo private

  say 'The resource ID of the Ok push button is:  ' self~newPushButton(IDOK)~getID
  say 'The identification number of this dialog is:' self~getID
```

4.10.23. getPos

```
>>--getPos---------------------------------------><
```

The getPos method returns the coordinates of the upper left corner a window, either a dialog or a dialog control, in dialog units. This method is not *inaccurate* because it uses the incorrect *factorX* and *factorY* attributes.

Return value:

The horizontal and vertical position, separated by a blank.

Example dialog control:

The following example repositions the tree view control FILES to the upper left corner of the window and displays the new position:

```
obj = MyDialog~newTreeView("FILES")
if obj = .Nil then return
obj~move(1,1)
parse value obj~getPos with x y
say "New horizontal position of window is" x "and new vertical position is" y
```

Example dialog:

The following example moves the window towards the top left of the screen.

```
parse value self~getPos with px py
self~move(px - 10, py - 10)
```

4.10.24. getRealPos

```
>>--getRealPos-----------------------------------><
```

Retrieves the position of the window in pixels.

Details

Sets the *.systemErrorCode*.

This method has no effect if the underlying dialog has not yet been created.

Arguments:

This method takes no arguments.

Return value:

The position of the window, in pixels, as a *Point* object. In the unlikely event of an error, the x and y coordinates will be 0. Of course those coordinates could be valid, check **`.systemErrorCode`** for a possible error code.

Example:

This example shrinks the dialog by 10 pixels:

```
::method move
  use strict arg newPosition

  pos = self~getRealPos
  say 'Current dialog position is ('pos~x','pos~y')'

  say 'Going to move to ('newPosition~x','newPosition~y') using moveTo()'
  self~moveTo(newPosition)
  say
  pos = self~getRealPos
  say 'getRealPos() says position is ('pos~x','pos~y')'
```

4.10.25. getRealSize

```
>>--getRealSize----------------------------------><
```

Retrives the size of the window in pixels.

Details

Sets the *.systemErrorCode*.

This method has no effect if the underlying dialog has not yet been created.

Arguments:

This method takes no arguments.

Return value:

The size of the window, in pixels, as a *Size* object. In the unlikely event of an error, the width and height will be 0.

Example:

This example shrinks the dialog by 10 pixels:

```
::method shrink

  size = self~getRealSize

  size~width -= 10
  size~height -= 10
  self~resizeTo(size)
```

4.10.26. getRect

```
>>--getRect(--+------+--)------------------------><
              +-hwnd-+
```

Retrieves the dimensions of the rectangle surrounding the associated dialog or dialog control. The coordinates are relative to the upper left corner of the screen and are specified in screen pixels. The order is: left, top, right, bottom; where 'left' and 'top' are the x and y coordinates of the upper left-hand corner of the rectangle, and 'right' and 'bottom' are the coordinates of the bottom right-hand corner.

Return value:

The coordinates of the dialog or dialog control, separated by blanks.

Example:

The following example calculates the width and height of an entry line:

```
parse value MyDialog~newEdit("Name")~getRect with left top,
right bottom
width = right - left
height = bottom - top
```

4.10.27. getSize

```
>>--getSize--------------------------------------><
```

The *getSize* method returns the width and height of the window in dialog units.

Details

The values returned by this method are usually *inaccurate* because they are calculated using *factorX* and *factorY*. The *getRealSize*() method returns the correct size in pixels. Normally it is not necessary to covert to dialog units. However, if this is desired, use the *pixel2dlgUnit* method to get the correct size.

This method can only be used after the *underlying* Windows dialog has been created.

Raises syntax errors for incorrect usage.

Arguments:

This method takes no arguments.

Return value:

A string with the width and height of the window, in dialog units, separated by a blank.

4.10.28. getStyleRaw

```
>>--getStyleRaw----------------------------------><
```

The *getStyleRaw* method retrieves the window style flags of the window. The flags are returned in their numeric format. This allows the programmer to determine the current style of any dialog or control. However, the programmer needs to be able to look up the numeric value of the window style flags to make use of this functionality. This can be done using the Windows *documentation* and the Platform *SDK*.

Details:

Using this method before the underlying dialog is created returns 0.

Arguments:

There are no arguments.

Return value:

The numeric value of the window style flags.

Example:

The following example gets the style flags for a tree control and then checks to see if drag and drop is enabled and if label editing is enabled.

The value of the style flags has the potential of being larger than can be accommodated by the default numeric digits. Use **`numeric digits 11`** if the returned value needs to be manipulated, for instance to use **`d2x`** on the value.

Note: it is not necessary to convert the *style* value in the example to its hexadecimal string representation before using it as an argument in the *and* method. That is just done in the example for display purposes.

```
...

numeric digits 11
TVS_DISABLEDRAGDROP = "0x0010"
TVS_EDITLABELS      = "0x0008"

treeView = self~newTreeView(IDC_TREE)
style = treeView~getStyleRaw
style = "0x" || style~d2x~right(8, '0')

if .DlgUtil~and( style, TVS_DISABLEDRAGDROP ) <> 0 then str1 = 'disabled'
else str1 = 'enabled'

if .DlgUtil~and( style, TVS_EDITLABELS ) <> 0 then str2 = 'enabled'
else str2 = 'disabled'

say 'Tree-view Control styles'
say '  Raw style:    ' style
say '  Drag and drop:' str1
say '  Edit labels:  ' str2
```

```
/* Output could be: */

Tree-view Control styles
  Raw style:     0x5000002F
  Drag and drop: enabled
  Edit labels:   enabled
```

4.10.29. getText

```
>>--getText--------------------------------------><
```

Gets the text associated with the window.

Details

Sets the *.systemErrorCode*.

Using this method before the underlying dialog is created will fail.

Arguments:

This method has no arguments.

Return value:

On success the text of the window, on error the empty string. However, the text of any window could be the empty string. Check **.systemErrorCode** for error, but an error is not likely.

Remarks:

All windows have text associated with them, even if it is the empty string. For top-level windows like dialogs it is the caption, or title of the dialog. For buttons it is the label for the button. For edit controls it is the text the user has entered, etc..

The *getText*, and *title* methods do exactly the same thing. The methods are interchangeable.

Example:

This example gets the text currently entered in an edit control and upper-cases it for uniformity.

```
::method upperDataField private
  use strict arg id

  edit = self~newEdit(id)
  text = edit~getText

  if text == "" then return .false

  edit~setText(text~upper)
  return .true
```

4.10.30. getTextSizePx

```
>>--getTextSizePx(-text--)-----------------------><
```

Gets the size, width and height, in pixels, needed to display a string in this window. To calculate the size in dialog units use the *getTextSizeDu*() method.

Details

This method can only be used after the *underlying* Windows dialog has been created.

Raises syntax errors for incorrect usage.

Arguments:

text

The string whose size is desired.

Return value:

The size needed for the string is returned in a *Size* object. The size is specified in pixels.

Example:

This example uses 2 static controls to display two strings. The static controls are positioned so that the second string immediately follows the first string. The size calculated is based on the font currently in use by the controls. The calculation is correct no matter what font is in use by the controls, even if the two controls use different fonts. (This is a contrived example, to illustrate the method. Instead of having a space character in between the 2 strings, the ending "t" and the beginning "a" are positioned with no space between them.)

```
stFont1 = self~newStaticText(IDC_ST_FONT1)
stFont2 = self~newStaticText(IDC_ST_FONT2)

-- Get the size needed for the message
size = stFont1~getTextSizePx("San Diego is great")
r = .Rect~new(1, 1, size~width, size~height)

stFont1~setText("San Diego is great")
stFont1~setRect(r)

size = stFont2~getTextSizePx("and ooRexx is the best")
r~left = 1 + r~right + 1
r~right = size~width
r~bottom = size~height

stFont2~setText("and ooRexx is the best")
stFont2~setRect(r)
```

4.10.31. getTextSizeScreen

```
>>--getTextSizeScreen(-text--+---------+--+-----------+--+-------------+-)----><
                             +-,-type--+  +-,-fontSrc--+  +-,-fontSize--+
```

Gets the size, width and height, in pixels, needed to display a string in a specific font.

Details

This method can only be used after the *underlying* Windows dialog has been created.

Raises syntax errors for incorrect usage.

Part of the purpose of this method is to provide backwards compatibility to the *deprecated* *getTextSize* method. This is the reason for the rather convoluted arguments. In general, the Rexx programmer should always use the *getTextSizePx* method.

Arguments:

The arguments are:

text [required]
: The string whose size is desired. If none of the optional arguments are specified then the font in use by this window (dialog or dialog control object) is used to calculate the size. This usage is exactly the same as using the *getTextSizePx* method.

type [optional / required]
: If any of the optional arguments are used this argument is required. It signals what *fontSrc* is. The allowed types are:

  Indirect
  : *fontSrc* is a font name and *fontSize* is the size of the font. The calculation is done indirectly by temporarily obtaining a logical font using these parameters.

  DC
  : *fontSrc* is a handle to a device context. The correct font for the calculation must already be selected into this device context. *fontSize* is ignored. See the *getDC*() and *fontToDC*() methods for information on obtaining a device context and selecting a font into it.

  Font
  : *fontSrc* is a handle to a font. *fontSize* is ignored.

  Only the first letter of type is needed and case is not significant.

fontSrc [optional]
: The source of the font to use in calculating the size. This argument object must match the *type* specification argument. It can be either a string, (a font name,) a handle to a device context, or a handle to a font.

fontSize [optional]
: The size of the font. This argument is always ignored unless the *type* argument is Indirect. If *type* is Indirect and this argument is omitted then the defualt font size is used. (Currently the default size is 8.)

Return value:

The size needed for the string is returned in a *Size* object. The size is specified in pixels.

4.10.32. hide

```
>>--hide-----------------------------------------><
```

The *hide* method makes the window invisible and sets the focus to the next window in the window order.

Details:

Using this method before the underlying dialog is created does nothing.

Arguments:

The method accepts no arguments.

Return value:

True if the window was previously visible, false if the window was previously hidden.

Example:

```
self~newEdit(IDC_EDIT_NAME)~hide
```

4.10.33. hideFast

```
>>--hideFast-------------------------------------><
```

The *hideFast* method marks the window as invisible but does not cause it to be redrawn.

Details:
Using this method before the underlying dialog is created does nothing.

Arguments:
The method accepts no arguments.

Return value:
0 for success, 1 on error. An error is extremely unlikely.

Remarks:
When a window is marked invisible, the operating system will not draw or redraw the window. It will only draw or redraw the area the window covered. For example if the window is part of another window's client area, and the client area is redrawn, then the window marked as invisible is not drawn. There are several methods that the programmer can use to cause a window to be redrawn, *update*, *draw*, etc..

To restate this for clarity, when the window is marked invisible, having the window update or redraw itself will do nothing. The window is hidden. Rather, the window beneath the hidden window must be redrawn. The consequence of this is, if the window marked invisible is a dialog control, then having the dialog update or redraw itself works well. However, if the window is a dialog, then it is the window beneath the dialog that must update. Normally the Rexx programmer will not have a way to force that window to update, and therefore the *hideFast* method does not work well for dialogs.

Example:

```
::method hideChoices private

  self~newGroupBox(IDC_GB_CHOICES)~hideFast
  self~newRadioButton(IDC_RB_CHOICE1)~hideFast
  self~newRadioButton(IDC_RB_CHOICE2)~hideFast
  self~newRadioButton(IDC_RB_CHOICE3)~hideFast
  self~newRadioButton(IDC_RB_CHOICE4)~hideFast
  self~newRadioButton(IDC_RB_CHOICE5)~hideFast
  self~newRadioButton(IDC_RB_CHOICE6)~hideFast
  ...
  self~update
```

4.10.34. isEnabled

```
>>--isEnabled------------------------------------><
```

Tests if the window is enabled

Details:

Using this method before the underlying dialog is created does nothing.

Arguments:

There are no arguments.

Return value:

Returns .true if the window is enabled, otherwise .false

Example:

```
::method toggleState private

listView = self~newListView(IDC_LV_NAMES)
if listView~isEnabled then
   listView~disable
else
   listView~enable
```

4.10.35. isVisible

```
>>--isVisible------------------------------------><
```

Tests if the window is visible.

Details:

Using this method before the underlying dialog is created does nothing.

Arguments:

There are no arguments.

Return value:

Returns true if the window is visible, otherwise false.

Example:

```

  if \ self~isVisible then self~show
```

4.10.36. mapWindowPoints

```
>>--mapWindowPoints(--hwndTo--,--points--)-------><
```

Converts, or maps, a set of points from the coordinate space relative to this window to a coordinate space relative to the specified window.

Arguments:

The arguments are:

hwndTo [required]

The window *handle* to map this window's coordinate space to.

points [required in / out]

The coordinates to map. This argument must be either a *Point* object or a *Rect* object. On a successful return, this object will be transformed so that it contains the mapped coordinates.

Return value:

Returns true if the mapping was successful, returns false on error.

Remarks:

This method is similar to the *client2screen* and *screen2client* methods except that the mapping is being done directly between 2 windows and the screen, the desktop window, is not involved. The *mapWindowPoints* methods has the same effect as would converting a point in this window's coordinate space to screen coordinate space and then converting that point to the *hwndTo* window's coordinate space.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example comes from a drag and drop program, where the user can drag an item from one list-view and drop it into some other list-view. The *mapWindowPoints* method is used to directly map points from / to the different list-view's coordinate space:

```
-- Take the West list-view's client area and map it to the
-- NFL list-view's coordinate space.

nfl2west = lvWest~clientRect
lvWest~mapWindowPoints(lvNFL~hwnd, nfl2west)

...

-- Some other portion of the program.  The point
-- p is in the NFL list-view's coordinate space.
-- We want to see if that point is over the West
-- list-view or over the East list-view.

if p~inRect(nfl2west) then do
  dragItem~target = lvWest
end
else if p~inRect(nfl2east) then do
  dragItem~target = lvEast
end
...

-- The point p is in client coordinates of the NFL list view.  We are
-- going to map that point back to the client coordinates of the list view
-- where the row is being dropped.  Then the drag item can determine where
-- the insertion point for the row should be in the target list view.

lvNFL~mapWindowPoints(dragItem~target~hwnd, p)
```

4.10.37. move

```
>>--move(--xPos--,--yPos--+-------------+--)-----><
                          +-,-showOpts--+
```

The move method moves the associated dialog or dialog control to the specified position. This method is *inaccurate* because it uses the incorrect *factorX* and *factorY* attributes.

Arguments:

The arguments are:

xPos

The new horizontal position of the dialog or dialog control, in dialog units.

yPos

The new vertical position of the dialog or dialog control, in dialog units.

showOptions

One or more of the following keywords, separated by blanks:

HIDEWINDOW

The dialog or dialog control is to be made invisible.

SHOWWINDOW

The dialog or dialog control is to be made visible.

NOREDRAW

The dialog or dialog control is to be repositioned without redrawing it.

Return value:

0

Moving was successful.

1

Moving failed.

Example:

The following example repositions the tree view control FILES to the upper left corner of the window and displays the new position:

```
obj = MyDialog~newTreeView("FILES")
if obj = .Nil then return
obj~move(1,1)
parse value obj~getPos with x y
say "New horizontal position of window is" x "and new vertical position is" y
```

4.10.38. moveTo

```
Form 1:

>>--moveTo(--point--+--------------+--)---------><
                    +--,-showOpts--+
Form 2:

>>--moveTo(--x,--y--+--------------+--)---------><
                    +--,-showOpts--+
Generic form:

>>--moveTo(--newPos--+--------------+--)--------><
                     +--,-showOpts--+
```

Moves, (repositions,) the window to the point specified. The new position is specified in pixels and is relative to the top, left corner of the owning window's *client area*. For a dialog this is the screen and for a dialog control it is the dialog's client area.

Arguments:

The first argument(s) specify the new position of the window. These coordinates can be specified using either a *Point* object, or by using separate x and y coordinates of the window.

newPos [required]

As noted above the new position for the window can be specified using a **`Point`** object or by specifying the separate x and y positions. If a point object is not used, both x and y are required.

showOpts [optional]

A list of 0 or one of the following keywords separated by spaces. Case is not significant. If the window is visible, this argument can, and probably should be, omitted. Combining the arguments makes little sense.

HIDEWINDOW	NOREDRAW
SHOWWINDOW	

HIDEWINDOW

The window is moved and made invisible.

SHOWWINDOW

The window is moved and, if it was hidden, it is made visible.

NOREDRAW

No repainting of any kind is done. The programmer is responsible for invalidating or redrawing any parts of the window and **parent** window that need redrawing. See the *redraw* method.

Return value:

The return values are:

true

Success.

false

Failure. Check **`.systemErrorCode`**.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

This method can not be used if the underlying dialog has not yet been created.

Example:

This example calculates a new position for a button control and then moves the button to that position. The button is originally created hidden. When it is moved to the correct position, it is then made visible.

```
::method initDialog

  revertButton = self~newPushButton(IDC_PB_REVERT)
  pos = self~calcRevertPosition
  revertButton~moveTo(pos, "SHOWWINDOW")
  ...
```

4.10.39. moveWindow

```
Form 1:

>>--moveWindow(--hwndBehind--,--point--+--------------+--)---------------------><
                                       +--,-showOpts--+
Form 2:

>>--moveWindow(--hwndBehind--,--x,--y--+--------------+--)---------------------><
                                       +--,-showOpts--+
Generic form:

>>--moveWindow(--hwndBehind--,--newPos--+--------------+--)--------------------><
                                        +--,-showOpts--+
```

Moves, (repositions,) the window to the point specified. The new position is specified in pixels and is relative to the top, left corner of the owning window's *client area*. For a dialog this is the screen and for a dialog control it is the dialog's client area.

Arguments:

The argument(s) after the *hwndBehind* argument specify the new position of the window. These coordinates can be specified using either a *Point* object, or by using separate x and y coordinates of the window.

hwndBehind [required]

The handle of a window that is to precede this window in the Z-order. This argument must be a valid window handle, or one of the following keywords. In addition, 0 is accepted. 0 is exactly equivalent to the TOP keyword.

BOTTOM	TOP
NOTTOPMOST	TOPMOST

BOTTOM

Places this window at the bottom of the Z order. If this window is a topmost window, it loses its topmost status and is placed at the bottom of all other windows.

NOTTOPMOST

Places this window above all non-topmost windows (that is, behind all topmost windows). This flag has no effect if this window is already a non-topmost window.

TOP

Places this window at the top of the Z order.

TOPMOST

Places this window above all non-topmost windows. This window maintains its topmost position even when it is deactivated.

newPos [required]

As noted above the new position for the window can be specified using a **`Point`** object or by specifying the separate x and y positions. If a point object is not used, both x and y are required.

showOpts [optional]

A list of 0 or more of the following keywords separated by spaces. Case is not significant. If this argument is omitted, the window is made visible and moved to the x and y coordinates specified. The NOSIZE keyword is automatically added to the options and does not need to be specified.

ASYNCWINDOWPOS	NOACTIVATE	NOSENDCHANGING
DEFERERASE	NOCOPYBITS	NOSIZE
DRAWFRAME	NOOWNERZORDER	NOZORDER
FRAMECHANGED	NOREDRAW	SHOWWINDOW
HIDEWINDOW	NOREPOSITION	

ASYNCWINDOWPOS
: If the calling thread and the thread that owns this window are attached to different input queues, the system posts the request to the thread that owns this window. This prevents the calling thread from blocking its execution while other threads process the request.

DEFERERASE
: Prevents generation of the WM_SYNCPAINT message.

DRAWFRAME
: Draws a frame (defined in this window's class description) around this window.

FRAMECHANGED
: Applies new frame styles set using the SetWindowLong function. Sends a WM_NCCALCSIZE message to this window, even if the window's size is not being changed. If this keyword is not specified, WM_NCCALCSIZE is sent only when the window's size is being changed.

HIDEWINDOW
: In addition to any moving or resizing that is done, this window is made invisible.

NOACTIVATE
: Does not activate this window. If this keyword is not used, this window is activated and moved to the top of either the topmost or non-topmost group (depending on the use of the hwndBehind argument.)

NOCOPYBITS
: Discards the entire contents of the client area. If this keyword is not specified, the valid contents of the client area are saved and copied back into the client area after this window is sized or repositioned.

NOOWNERZORDER
: Does not change the owner window's position in the Z order.

NOREDRAW
: No repainting of any kind is done. The programmer is responsible for invalidating or redrawing any parts of the window and **parent** window that need redrawing. See the *redraw* method.

NOREPOSITION
: Same as the NOOWNERZORDER keyword.

NOSENDCHANGING
: Prevents this window from receiving the WM_WINDOWPOSCHANGING message.

NOSIZE
: This keyword is added automatically and does not need to be specified. However, it does no harm to use it.

NOZORDER
: Retains the current Z order (ignores the hwndBehind parameter).

SHOWWINDOW
In addition to any moving or resizing that is done, this window is made visible.

Return value:

The return values are:

true
Success.

false
Failure. Check **`.systemErrorCode`**.

Details

This method provides an interface to the *SetWindowPos* Windows API. Consulting the Microsoft *documentation* may give the programmer more insight into this method. The *moveTo*() method is a convenience method that provides a somewhat simplier interface. That convenience method has no provision for the *hwndBehind* argument and only accepts a subset of the show options.

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

This method can not be used if the underlying dialog has not yet been created.

Example:

This example calculates the position a number of **`ControlDialogs`** need so that they overlap the display area in a tab control and then moves them all to that position. When the dialog appears, the first tab is displayed, so the control dialog for that tab is made visible while the other control dialogs remain hidden. Since the NOZORDER is not used, all the control dialogs are placed so that they are ahead of the tab control window. This ensures that the tab control is displayed properly.

```
::method initDialog
  expose controlDlgs

  ...

  newPos = self~calcPosition(controlDlgs)

  do i = 1 to controlDlgs~items
    dlg = controlDlgs[i]
    if i == 1 then dlg~moveWindow(tab~hwnd, newPos, "SHOWWINDOW")
    else dlg~moveWindow(tab~hwnd, newPos)
  end

  ...
```

4.10.40. redraw

```
>>--redraw---------------------------------------><
```

The *redraw* method immediately redraws the entire window and all its child windows, recursively, including the non-*client area* of the window(s). Contrast this with some of the other draw methods such as *redrawClient* or *draw*, which only redraw the *client* area and not the child windows.

Details:

Using this method before the underlying dialog is created does nothing.

Sets the *.systemErrorCode*.

Arguments:

There are no arguments.

Return value:

The return values are:

0
: Redrawing was successful.

1
: Redrawing failed. Check **`.systemErrorCode`**.

4.10.41. redrawClient

```
>>--redrawClient(--+------------+--)-------------><
                   +--eraseBkg--+
```

The *redrawClient* method causes the entire *client area* of the window to be redrawn.

Details:

Using this method before the underlying dialog is created does nothing.

Sets the *.systemErrorCode*.

Arguments:

The only argument is:

eraseBkg [optional]
: Whether the background of the window should be erased before redrawing. If you specify this argument and it is true or "Y", the background of the dialog is erased before redrawing. The default if the argument is omitted is false.

Return value:

The return values are:

0
: Redrawing was successful.

1
: Redrawing failed. Check **`.systemErrorCode`**.

Remarks:

Having a window redraw its client area has no effect if the window itself is hidden, there is nothing to draw. On the other hand, if a window, such as a dialog window, contains hidden windows that are still showing, then having it redraw has the effect of erasing the hidden windows, *provided* that the background is first erased.

4.10.42. resize

```
>>--resize(--width--,--height--+------------+--)---------------><
```

```
                              +-,-showOpts--+
```

The *resize* method resizes a dialog or dialog control window. This old method is not *accurate*.

Arguments:

The arguments are:

width [required]

The new width of the window, in dialog units.

height [required]

The new height of the window, in dialog units.

showOptions [optional]

A list of 0 or more of the following keywords separated by spaces. Case is not significant:

HIDEWINDOW	SHOWWINDOW	NOREDRAW

HIDEWINDOW

The window is resized and hidden.

SHOWWINDOW

The window is resized and made visible.

NOREDRAW

The window is resized, but is not forced to redraw.

Return value:

The possible return values are:

0

Resizing was successful.

1

Resizing failed.

Remarks

A window can only be resized once the underlying window has first been created. The size of the window is then specified in pixels. What this method does, is take the input values in dialog units and convert them to pixels. Since the conversion is done using *factorX* and *factorY*, the conversion is not correct.

When working with windows, it is simplier and easier to stick with pixels. To obtain accurate resizing of a window, use the *resizeTo* and *getRealSize* methods. Or, the *setRect* and *windowRect* methods.

Example:

The following example resizes the tree view control with resource id of IDC_TV_FILES almost to the size of the dialog window, and then prints out the new size:

```
obj = self~newTreeView("FILES")
if obj = .nil then return
obj~resize(self~sizeX - 10, self~sizeY - 20)
parse value obj~getSize with width height
say "New width of tree view is" width "and new height is" height
```

4.10.43. resizeTo

```
Form 1:

>>--resizeTo(--size--+--------------+--)---------><
                     +--,-showOpts--+

Form 2:

>>--resizeTo(--cx,--cy--+--------------+--)------><
                        +--,-showOpts--+

Generic form:

>>--resizeTo(--newSize--+--------------+--)------><
                        +--,-showOpts--+
```

Resizes the window to that specified. The new size is specified in pixels.

Arguments:

The first argument(s) specify the new size of the window. These coordinates can be specified using either a *Size* object, or by the new width and new height of the window.

newSize [required]

As noted above the new size for the window can be specified using a **`Size`** object or by specifying the new width and then the new height. Both the width and the height are required.

showOpts [optional]

A list of 0 or one of the following keywords separated by spaces. Case is not significant. If the window is visible, this argument can, and probably should be, omitted. Combining the arguments makes little sense.

HIDEWINDOW	SHOWWINDOW	NOREDRAW

HIDEWINDOW

The window is resized and made invisible.

SHOWWINDOW

The window is resized and, if it was hidden, it is made visible.

NOREDRAW

No repainting of any kind is done. The programmer is responsible for invalidating or redrawing any parts of the window and **parent** window that need redrawing. See the *redraw* method.

Return value:

The return values are:

true

Success.

false

Failure. Check **`.systemErrorCode`**.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

This method can not be used if the underlying dialog has not yet been created.

Example:

This example gets the minimum size needed for a month calendar control and resizes the control to the result.

```
::method initDialog

  calendar = self~newMonthCalendar(IDC_MC_VACATION)
  r = .Rect~new

  calendar~getMinRect(r)
  s = .Size~new(r~right, r~bottom)
  calendar~resizeTo(s)
  ...
```

4.10.44. screen2client

```
Form 1:

>>--screen2client(--point--)---------------------><

Form 2:

>>--screen2client(--rect--)----------------------><

Generic form:

>>--screen2client(--obj--)-----------------------><
```

Converts the screen coordinates of the specified point(s) on the screen to *client area* coordinates of this window.

Arguments:

The single argument specifies the screen coordinates to be converted. It must be either a **`.Point`** or a **`.Rect`** object. A **`.Rect`** can be used to specify two points. Point one is taken to be the **`left`** and **`top`** attributes and point two is taken to bbe the **`right`** and **`bottom`** attributes.

obj [required in/out]

The point(s) to be converted.

Return value:

True on success, false on failure.

Remarks:

This method uses the screen coordinates given in the *obj* argument to compute client-area coordinates. It then replaces the screen coordinates with the client coordinates. The new coordinates are relative to the upper-left corner of this window's client area. Coordinates are expressed in pixels.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example gets the position of a tab control in screen coordinates and converts the position to its position in the client area of the dialog:

```
tabControl = self~newTab(IDC_TAB)
r = tabControl~windowRect

p = .Point~new(r~left, r~top)
self~screen2client(p)

say "The tab control is positioned at: ("p~x"," p~y") in the dialog's client area."
```

4.10.45. screenToClient

```
>>--screenToClient(--x--,--y--)------------------><
```

The *screenToClient* method maps the screen coordinates specified to the *client area* of this window.

Arguments:

The arguments are:

x [required]

The horizontal position, in screen pixels.

y [required]

The vertical position, in screen pixels.

Return value:

The client-area coordinates of the specified screen location, separated by a blank.

Remarks:

This old ooDialog method returns the coordinates as a single string that needs to be parsed. This is often not that convenient. The *screen2client* method is often easier to use.

4.10.46. sendMessage

```
>>--sendMessage(--id--,--msg--,--wParam--,--lParam--)--------------------------><
```

Sends a Windows message to the underlying window this object represents and returns the response as a whole number.

Arguments:

The arguments are:

msg [required]

The Windows message ID. This may be numeric or a string in conventional *hexadecimal* format.

wParam [required]

The WPARAM message parameter.

lParam [required]

The LPARAM message parameter.

Return value:

The underlying windows's response to the message, as a whole number.

Remarks:

This method returns the response as a whole number. This will not be usable if the response is a window *handle*. For those cases, use the *sendMessageHandle*() method.

It is not possible in ooRexx to send a message directly to a window when either the WPARAM or LPARAM arguments are strings. For those cases use a suitable method in one of the classes provided by the ooDialog framework.

Details:

Sets the *.systemErrorCode*.

This method requires an understanding of Windows *message*s.

This method can not be used before the underlying dialog is created.

Example:

The following example causes the dialog to reposition itself that that it completely fits within the desktop area. (The *ensureVisible*() method does the same thing.)

```
MovableDialog~sendMessage"0x0402", 0, 0)
```

4.10.47. sendMessageHandle

```
>>--sendMessageHandle(--id--,--msg--,--wParam--,--lParam--)--------------------><
```

Sends a Windows message to the underlying window this object represents and returns the response as a *handle*.

Arguments:

The arguments are:

msg [required]

The Windows message ID. This may be numeric or a string in conventional *hexadecimal* format.

wParam [required]

The WPARAM message parameter.

lParam [required]

The LPARAM message parameter.

Return value:

The underlying windows's response to the message, as a handle

Remarks:

This method returns the response as a handle. This will not be usable if the response is a whole number. For those cases, use the *sendMessage*() method.

It is not possible in ooRexx to send a message directly to a window when either the WPARAM or LPARAM arguments are strings. For those cases use a suitable method in one of the classes provided by the ooDialog framework.

Details:

Sets the *.systemErrorCode*.

This method requires an understanding of Windows *message*s.

This method can not be used before the underlying dialog is created.

Example:

The following example sets a new *dialog icon*). The old icon handle is saved so it can be restored at some later point.

```
::method initDialog
  expose oldIcon

  oldIcon = .nil
  ...

::method swapIcon private
  expose oldIcon
  use strict arg iconID

  loadFlags = .DlgUtil~or(.Image~toID(LR_DEFAULTSIZE), .Image~toID(LR_LOADFROMFILE))
  customIcon = .Image~userIcon(self, IDI_SHAVEICE, .Size~new(0, 0), loadFlags)

  if \ customIcon~isNull then do
    oldIcon = self~sendMessageHandle("0x0080", 0, customIcon~handle)
    return .true
  end

  return .false

::method restoreAppState private
  expose oldIcon

  if oldIcon <> .nil then self~sendMessageHandle("0x0080", 0, oldIcon)
```

4.10.48. setRect

```
Form 1:

>>--setRect(--rectangle--+------------+--)------->< 
                         +-,-showOpts-+

Form 2:

>>--setRect(--point--,--size--+------------+--)-->< 
                              +-,-showOpts-+

Form 3:

>>--setRect(--x-,--y-,--cx-,--cy--+------------+--)------------->< 
                                  +-,-showOpts-+

Generic form:

>>--setRect(--ptSizeRectangle--+------------+--)->< 
                               +-,-showOpts-+
```

Sets the new *point / size* rectangle for the window. A point / size rectangle specifies the position of the upper left corner of the window, relative to its parent window, the width of the window, and the height of the window. The point / size rectangle coordinates are specified in pixels.

In essence this moves and resizes the window. By using different *showOpts*, the window can be only moved or only resized.

Arguments:

The first argument(s) specify the new coordinates of the window. These coordinates can be specified using either a *Rect* object, or by using a *Point* object followed by a *Size* object, or by using the x, y coordinates of the upper left point of the window followed by the width and height of the window.

ptSizeRectangle [required]

The point / size rectangle argument(s) are all required. They specify the left and top position of the window, the width, and height of the window. As noted above, these coordinates can be specified in the format that is most convenient.

Note carefully: If a **`Rect`** object is used to specify the point / size rectangle, the semantics are not quite correct. The *right* member of the **`Rect`** must contain the width of the window and the *bottom* member must contain the height of the window.

When the *showOpts* argument contains the NOMOVE keyword, the x, y coordinates are ignored. Likewise, if *showOpts* contains NOSIZE, the width and height portion of the point / size rectangle is ignored.

showOpts [optional]

A list of 0 or more of the following keywords separated by spaces. Case is not significant. If this argument is omitted, the window is made visible, moved to the position specified by the x, y coordinates and resized to the width and height specified by the width and height portion of the point / size rectangle

NOSIZE	HIDEWINDOW	NOREDRAW
NOMOVE	SHOWWINDOW	

NOSIZE

The window will not be resized. The new width and height specified is ignored. The values can be 0, or any other value.

NOMOVE

The position of the window will not be changed. The new x, y position is ignored. Again, the values can be 0, or any other value.

HIDEWINDOW

In addition to any moving or resizing that is done, the window is made invisible.

SHOWWINDOW

In addition to any moving or resizing that is done, the window is made visible.

NOREDRAW

No repainting of any kind is done. The programmer is responsible for invalidating or redrawing any parts of the window and **parent** window that need redrawing. See the *redraw* method.

Return value:

The return values are:

true

Success.

false

Failure. Check **`.systemErrorCode`**.

Remarks:

Note that the Microsoft *documentation* says that if HIDEWINDOW or SHOWWINDOW is used, the window can not be moved or resized. However, experimentation seems to show that the documentation is wrong. It may be that this behavior is different on different versions of Windows.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

This method can not be used if the underlying dialog has not yet been created.

Example:

This example increases the size and width of the dialog by 20 pixels and moves the position of the dialog up and to the left by 10 pixels. This has the effect of enlarging the dialog while keeping the center of the dialog at the same spot.

```
::method zoomOut private

 rect = self~windowRect
 rect~right = rect~right - rect~left + 20
 rect~bottom = rect~bottom - rect~top + 20

 rect~left -= 10
 rect~top -= 10

 self~setRect(rect)
```

4.10.49. setStyleRaw

```
>>--setStyleRaw(--value--)-----------------------><
```

The *setStyleRaw* method sets the window style flags of the window. The value specified is the raw numeric value. This allows the programmer to set the current style of any dialog or control. However, the programmer needs to be able to look up the numeric value of the window style flags to make use of this functionality. This can be done using the Windows *documentation* and the Platform *SDK*.

Arguments:

The single argument is:

value [required]

The new value for the window's style flags. This can either be a non-negative whole number, or for convenience, the number can be expressed as a string in conventional *hexadecimal* format.

Return value:

The old value for the window style flags. That is, the numeric value of the style flags before the new value was applied.

Remarks:

There is no way that the ooDialog framework can detect if the value supplied by the programmer is valid for the window. The programmer is responsible for ensuring the value is correct. Using a value not documented by Microsoft as valid, will produce undefined results.

Details:

Raises syntax errors when incorrect usage is detected.

Using this method before the underlying dialog is created returns 0.

Example:

The following example gets the current style flags for a list-view control and then adds the WS_CLIPCHILDREN style to the list-view window. (WS_CLIPCHILDREN equals 0x02000000.) The results are then printed to the screen.

The value of the style flags has the potential of being larger than can be accommodated by the default numeric digits. Use **`numeric digits 11`** if the returned value needs to be manipulated, for instance to use **`d2x`** on the value.

```
  ...

  numeric digits 11

  old = list~setStyleRaw(.DlgUtil~or(list~getStyleRaw, "0x02000000"))
  new = list~getStyleRaw

  old = "0x" || old~d2x~right(8, '0')
  new = "0x" || new~d2x~right(8, '0')

  say 'Old list-view style:' old
  say 'New list-view style:' new


/*  Output might be for instance: */

  Old list-view style: 0x50010009
  New list-view style: 0x52010009
```

4.10.50. setText

```
>>--setText(--newText--)-------------------------><
```

Sets the text associated with the window.

Details

Sets the *.systemErrorCode*.

Using this method before the underlying dialog is created will fail.

Arguments:

The single argument is:

newText [required]

The new text for the window.

Return value:

The return values are:

0
: Success.

1
: Failure. Check `.systemErrorCode`

Remarks:

All windows have text associated with them, even if it is the empty string. For top-level windows like dialogs it is the caption, or title of the dialog. For buttons it is the label for the button. For edit controls it is the text the user has entered, etc..

The *setText*, *setTitle*, and *title=* methods do exactly the same thing. The methods are interchangeable.

Example:

This example sets the text that a static text control displays:

```
  ::method reportError private
    use strict arg errMsg
    self~newStatic(IDC_ST_INFO)~setText(errMsg)
```

4.10.51. setTitle

```
>>--setTitle(--newText--)------------------------><
```

The *setTitle* method is an alias for *setText*().

4.10.52. setWindowPos

```
Form 1:

>>--setWindowPos(--hwndBehind--,--rectangle--+------------+--)-----------------><
                                             +-,-showOpts-+

Form 2:

>>--setWindowPos(--hwndBehind--,--point--,--size--+------------+--)------------><
                                                  +-,-showOpts-+

Form 3:

>>--setWindowPos(--hwndBehind--,--x-,--y-,--cx-,--cy--+------------+--)--------><
                                                      +-,-showOpts-+

Generic form:

>>--setWindowPos(--hwndBehind--,--ptSizeRectangle--+------------+--)-----------><
                                                   +-,-showOpts-+
```

The *setWindowPos* method is used to change the size, position, visibility, and Z order of this window. Windows are ordered according to their appearance on the screen. The topmost window receives the highest rank and is the first window in the Z order.

Arguments:

The argument(s) after the *hwndBehind* argument specify the new point / size rectangle of the window. A point / size rectangle specifies the position of the upper left corner of the window, relative to its parent window, the width of the window, and the height of the window. The point / size rectangle is specified in pixels.

The arguments can be specified using either a *Rect* object, or by using a *Point* object followed by a *Size* object, or by using the x, y coordinates of the upper left point of the window followed by the width and height of the window.

hwndBehind [required]

The handle of a window that is to precede this window in the Z-order. This argument must be a valid window handle, or one of the following keywords. In addition, 0 is accepted. 0 is exactly equivalent to the TOP keyword.

BOTTOM	TOP
NOTTOPMOST	TOPMOST

BOTTOM

Places this window at the bottom of the Z order. If this window is a topmost window, it loses its topmost status and is placed at the bottom of all other windows.

NOTTOPMOST

Places this window above all non-topmost windows (that is, behind all topmost windows). This flag has no effect if this window is already a non-topmost window.

TOP

Places this window at the top of the Z order.

TOPMOST

Places this window above all non-topmost windows. This window maintains its topmost position even when it is deactivated.

ptSizeRectangle [required]

The point / size rectangle argument(s) are all required. They specify the new left and top position of this window, the width, and height of this window. As noted above, these coordinates can be specified in the format that is most convenient.

Note carefully: If a **`Rect`** object is used to specify the point / size rectangle, the semantics are not quite correct. The *right* member of the **`Rect`** must contain the width of the window and the *bottom* member must contain the height of the window.

When the *showOpts* argument contains the NOMOVE keyword, the x, y coordinates are ignored. Likewise, if *showOpts* contains NOSIZE, the width and height portion of the point / size rectangle is ignored.

showOpts [optional]

A list of 0 or more of the following keywords separated by spaces. Case is not significant. If this argument is omitted, the window is made visible, moved to the position specified by the x, y coordinates and resized to the width and height specified by the width and height portion of the point / size rectangle

ASYNCWINDOWPOS	NOACTIVATE	NOREPOSITION
DEFERERASE	NOCOPYBITS	NOSENDCHANGING
DRAWFRAME	NOMOVE	NOSIZE
FRAMECHANGED	NOOWNERZORDER	NOZORDER
HIDEWINDOW	NOREDRAW	SHOWWINDOW

ASYNCWINDOWPOS

If the calling thread and the thread that owns this window are attached to different input queues, the system posts the request to the thread that owns this window. This prevents the calling thread from blocking its execution while other threads process the request.

DEFERERASE

Prevents generation of the WM_SYNCPAINT message.

DRAWFRAME

Draws a frame (defined in this window's class description) around this window.

FRAMECHANGED

Applies new frame styles set using the SetWindowLong function. Sends a WM_NCCALCSIZE message to this window, even if the window's size is not being changed. If this keyword is not specified, WM_NCCALCSIZE is sent only when the window's size is being changed.

HIDEWINDOW

In addition to any moving or resizing that is done, this window is made invisible.

NOACTIVATE

Does not activate this window. If this keyword is not used, this window is activated and moved to the top of either the topmost or non-topmost group (depending on the use of the hwndBehind argument.)

NOCOPYBITS

Discards the entire contents of the client area. If this keyword is not specified, the valid contents of the client area are saved and copied back into the client area after this window is sized or repositioned.

NOMOVE

The position of this window will not be changed. The new x, y position is ignored. The x, y values can be 0, or any other value.

NOOWNERZORDER

Does not change the owner window's position in the Z order.

NOREDRAW

No repainting of any kind is done. The programmer is responsible for invalidating or redrawing any parts of the window and **parent** window that need redrawing. See the *redraw* method.

NOREPOSITION

Same as the NOOWNERZORDER keyword.

NOSENDCHANGING

Prevents this window from receiving the WM_WINDOWPOSCHANGING message.

NOSIZE

This window will not be resized. The new width and height specified is ignored. The width and height values can be 0, or any other value.

NOZORDER

Retains the current Z order (ignores the hwndBehind parameter).

SHOWWINDOW

In addition to any moving or resizing that is done, this window is made visible.

Return value:

The return values are:

true

Success.

false

Failure. Check **`.systemErrorCode`**.

Remarks:

This method provides a complete interface to the *SetWindowPos* Windows API. Consulting the Microsoft *documentation* may give the programmer more insight into this method. The *setRect*() method is a convenience method that provides a somewhat simplier interface. That convenience method has no provision for the *hwndBehind* argument and only accepts a subset of the show options.

Note that the Microsoft *documentation* says that if HIDEWINDOW or SHOWWINDOW is used, the window can not be moved or resized. However, experimentation seems to show that the documentation is wrong. It may be that this behavior is different on different versions of Windows.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

This method can not be used if the underlying dialog has not yet been created.

Example:

This example increases the size and width of the dialog by 20 pixels and moves the position of the dialog up and to the left by 10 pixels. This has the effect of enlarging the dialog while keeping the center of the dialog at the same spot.

```
::method zoomOut private

 rect = self~windowRect
 rect~right = rect~right - rect~left + 20
 rect~bottom = rect~bottom - rect~top + 20

 rect~left -= 10
 rect~top -= 10

 self~setWindowPos(TOP, rect)
```

4.10.53. show

```
>>--show-----------------------------------------><
```

Makes the window visible in its normal size and position. For a dialog control window, this simply makes it visible, if it was previously hidden. A dialog window is also activated, (given the focus for all intents and purposes,) and restored from a minimized or maximized state if needed.

Details:

Using this method before the underlying dialog is created does nothing.

Arguments:

There are no arguments.

Return value:

Returns true if the window was previously visible and false if the window was previously hidden.

Example:

The following example makes the button controls visible:

```
::method showButtons private

  self~newPushbutton(IDC_PB_COMMIT)~show
  self~newPushbutton(IDC_PB_SUBMIT)~show
  self~newPushbutton(IDC_PB_REVERT)~show
```

4.10.54. showFast

```
>>--showFast-------------------------------------><
```

Marks the window as visible but does not cause it to be redrawn.

Details:

Using this method before the underlying dialog is created does nothing.

Arguments:

The method accepts no arguments.

Return value:

0 for success, 1 on error. An error is extremely unlikely.

Remarks:

When a window is marked visible, the operating system does not redraw the window until necessary. For example if the window is covered by another window and then uncovered. There are several methods that the programmer can use to cause a window to be redrawn, *update*), *draw*, etc..

Example:

```
::method showChoices private

  self~newGroupBox(IDC_GB_CHOICES)~showFast
  self~newRadioButton(IDC_RB_CHOICE1)~showFast
  self~newRadioButton(IDC_RB_CHOICE2)~showFast
  self~newRadioButton(IDC_RB_CHOICE3)~showFast
  self~newRadioButton(IDC_RB_CHOICE4)~showFast
  self~newRadioButton(IDC_RB_CHOICE5)~showFast
  self~newRadioButton(IDC_RB_CHOICE6)~showFast
  ...
  self~update
```

4.10.55. sizeWindow

```
Form 1:
```

```
>>--sizeWindow(--hwndBehind--,--size--+-------------+--)------><
                                      +--,-showOpts--+

Form 2:

>>--sizeWindow(--hwndBehind--,--cx,--cy--+-------------+--)----><
                                         +--,-showOpts--+

Generic form:

>>--sizeWindow(--hwndBehind--,--newSize--+-------------+--)----><
                                         +--,-showOpts--+
```

Resizes the window to the size specified. The new size is specified in pixels.

Arguments:

The argument(s) after the *hwndBehind* argument specify the new size of the window. These coordinates can be specified using either a *Size* object, or by the new width and new height of the window.

hwndBehind [required]

The handle of a window that is to precede this window in the Z-order. This argument must be a valid window handle, or one of the following keywords. In addition, 0 is accepted. 0 is exactly equivalent to the TOP keyword.

BOTTOM	TOP
NOTTOPMOST	TOPMOST

BOTTOM

Places this window at the bottom of the Z order. If this window is a topmost window, it loses its topmost status and is placed at the bottom of all other windows.

NOTTOPMOST

Places this window above all non-topmost windows (that is, behind all topmost windows). This flag has no effect if this window is already a non-topmost window.

TOP

Places this window at the top of the Z order.

TOPMOST

Places this window above all non-topmost windows. This window maintains its topmost position even when it is deactivated.

newSize [required]

As noted above the new size for the window can be specified using a `Size` object or by specifying the new width and then the new height. Both the width and the height are required.

showOpts [optional]

A list of 0 or more of the following keywords separated by spaces. Case is not significant. If this argument is omitted, the window is made visible and resized to the width and height specified. The NOMOVE keyword is automatically added to the options and does not need to be specified.

ASYNCWINDOWPOS	NOACTIVATE	NOREPOSITION
DEFERERASE	NOCOPYBITS	NOSENDCHANGING
DRAWFRAME	NOMOVE	NOZORDER
FRAMECHANGED	NOOWNERZORDER	SHOWWINDOW
HIDEWINDOW	NOREDRAW	

ASYNCWINDOWPOS
: If the calling thread and the thread that owns this window are attached to different input queues, the system posts the request to the thread that owns this window. This prevents the calling thread from blocking its execution while other threads process the request.

DEFERERASE
: Prevents generation of the WM_SYNCPAINT message.

DRAWFRAME
: Draws a frame (defined in this window's class description) around this window.

FRAMECHANGED
: Applies new frame styles set using the SetWindowLong function. Sends a WM_NCCALCSIZE message to this window, even if the window's size is not being changed. If this keyword is not specified, WM_NCCALCSIZE is sent only when the window's size is being changed.

HIDEWINDOW
: In addition to any moving or resizing that is done, this window is made invisible.

NOACTIVATE
: Does not activate this window. If this keyword is not used, this window is activated and moved to the top of either the topmost or non-topmost group (depending on the use of the hwndBehind argument.)

NOCOPYBITS
: Discards the entire contents of the client area. If this keyword is not specified, the valid contents of the client area are saved and copied back into the client area after this window is sized or repositioned.

NOMOVE
: This argument is added automatically, it does not need to be used. However, it does no harm if it is used.

NOOWNERZORDER
: Does not change the owner window's position in the Z order.

NOREDRAW
: No repainting of any kind is done. The programmer is responsible for invalidating or redrawing any parts of the window and **parent** window that need redrawing. See the *redraw* method.

NOREPOSITION
: Same as the NOOWNERZORDER keyword.

NOSENDCHANGING
: Prevents this window from receiving the WM_WINDOWPOSCHANGING message.

NOZORDER
: Retains the current Z order (ignores the hwndBehind parameter).

SHOWWINDOW
: In addition to any moving or resizing that is done, this window is made visible.

Return value:

The return values are:

true

Success.

false

Failure. Check **`.systemErrorCode`**.

Details

This method provides an interface to the *SetWindowPos* Windows API. Consulting the Microsoft *documentation* may give the programmer more insight into this method. The *resizeTo*() method is a convenience method that provides a somewhat simplier interface. That convenience method has no provision for the *hwndBehind* argument and only accepts a subset of the show options.

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

This method can not be used if the underlying dialog has not yet been created.

Example:

This example gets the minimum size needed for a month calendar control and resizes the control to the result.

```
::method initDialog

  calendar = self~newMonthCalendar(IDC_MC_VACATION)
  r = .Rect~new

  calendar~getMinRect(r)
  s = .Size~new(r~right, r~bottom)
  calendar~sizeWindow(TOP, s)
  ...
```

4.10.56. title

```
>>--title----------------------------------------><
```

The *title* method is an alias for *getText*().

4.10.57. title=

```
>>--title=newText--------------------------------><
```

The *title*= method is an alias for *setText*().

4.10.58. update

```
>>--update---------------------------------------><
```

The *update* method causes the entire *client area* of the window to be invalidated and the background erased. This method is similar to both the *redrawClient* and *draw* methods except that the window is, potentially, not repainted immediately.

Details:

Using this method before the underlying dialog is created does nothing.

Sets the *.systemErrorCode*.

Arguments:

There are no arguments for this method.

Return value:

The return values are:

0

The invalidation was successful.

1

Invalidation failed. Check **`.systemErrorCode`**.

Remarks:

Invalidating an area of a window, in this case the entire client area, will cause that area to be repainted at some point in the future. When an area of a window is invalid, the operating system will send a message to the window to repaint that area, *when there are no other messages* in the window's message queue. In effect, the area invalidated is *scheduled* to be redrawn when the window is no longer busy.

Scheduling a window to redraw its client area has no effect if the window itself is hidden, there is nothing to draw. On the other hand, if a window, such as a dialog window, contains hidden windows that are still showing, then having it redraw has the effect of erasing the hidden windows, *provided* that the background is first erased.

4.10.59. updateWindow

```
>>--updateWindow---------------------------------><
```

Updates the client area of this window by sending a paint message to the window, if the window's update region is not empty.

Arguments:

There are no arguments to this method.

Return value:

Returns true on success, false on error.

Remarks:

Normally paint messages are added to the application message queue by the operating system only when the application queue is empty. The window processes this messages when it is not busy. The *updateWindow* method causes the operarating system to send the paint message directly to the message processing procedure of the window, bypassing the application queue. The result of this is that any invalid part of the window is redrawn immediately.

Note that if there is no invalid portion of the window, this method has no effect. The operating system simply does nothing.

Details

Raises syntax errors when incorrect usage is detected. This method can not be used if the underlying dialog does not exist.

Sets the *.SystemErrorCode* variable.

Example:

In this example, the list view is in report mode. The code snippet causes the list view to invalidate 3 rows in the view. Normally the 3 rows would not be redrawn until the list view received the next paint message. Using the *updateWindow* method causes the 3 rows to be redrawn immediately:

```
if keyEvent == 'none' then do
    listView~redrawItems(currentIndex - 1, currentIndex + 1)
    listView~updateWindow
end
```

4.10.60. windowRect

```
>>--windowRect(--+--------+--)-------------------><
                 +--hwnd--+
```

Retrieves the coordinates of the bounding *rectangle* of this window. The dimensions are in pixels and the coordinates are relative to the upper-left corner of the screen.

Arguments:

The single argument is:

hwnd [optional]

By default, the coordinates are for this window. However, the optional *hwnd* argument can be used to specify getting the coordinates for some other window.

Return value:

Returns the bounding rectangle of the window as a *Rect* object.

Remarks:

The *windowRect* method supplies an alternative to the *getRect*(hwnd) method, to allow returning a **`.Rect`** object, which is usually more convenient, rather than a string of blank separated values.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example uses *windowRect* to get the position of a tab control on the screen and then translates those coordinates to its position in the *client area* of the dialog.

```
tabControl = self~newTab(IDC_TAB)
r = tabControl~windowRect

p = .Point~new(r~left, r~top)
self~screen2client(p)

say "The tab control is positioned at: ("p~x"," p~y") in the dialog's client area."
```

4.11. WindowExtensions Mixin Class

WindowExtensions is a mixin class with methods that *should* be common to all windows. The class name implies it is meant to be an extension of the *WindowBase* class, providing more sophisticated window methods. It is inherited by both the *dialog* and the dialog *control* objects.

Unfortunately, **many** of the methods placed in the WindowExtensions class were not really *window* methods. Rather they were extensions to the ooDialog framework. Otherwise in the *effort* to simplify ooDialog all the methods would have been moved to the WindowBase class and the WindowExtensions class would have been eliminated. However, the methods remaining in WindowExtensions, do not belong in a *window* class, so that simplification could not be done.

In addition, most of the methods do not really belong to a *dialog control* specific class. But, since historically the dialog control did have these methods, they need to remain a part of the dialog control object for backwards compatibility. Therefore, both the name of the class and the usage of its methods from a dialog control object are not really appropriate.

4.11.1. Method Table

The following table lists the instance methods of the `WindowExtensions` Mixin Class.

Table 4.14. WindowExtensions Method Reference

Method	Description
Instance Methods	**Instance Methods**
createBrush	Creates a logical brush that has the specified style, color, and pattern.
createFont	Returns a handle to a logical font, the implementation is **incorrect**.
createPen	Creates a logical pen that has the specified style, width, and color.
createFontEx	Retrieves a handle to a logical font from the system font manager
deleteFont	Deletes a font returned from *createFontEx* or *createFont*.
deleteObject	Deletes a graphic object,
drawAngleArc	Draws a partial circle (arc) and a line connecting the start drawing point with the start of the arc.
drawArc	Draws a circle or ellipse withi the given device context using the active pen.
drawLine	Draws a line within the device context using the active pen.
drawPie	Draws and fills a pie of a circle or ellipse within the given device context using the active brush and pen.
drawPixel	Draws a pixel within the device context.
fillDrawing	Fills an area of the display surface with the current brush. The area to be filled is outlined by the color specified.
fillRect	Fills a rectangle using the specified brush within the specified device context.
fontColor	Sets the font color for a device context.
fontToDC	Loads a font into a device context and returns the handle of the previous font.
freeDC	Releases a device context.
getArcDirection	Returns the current drawing direction.
getFont	Returns the font used by the dialog.
getPixel	Returns the color number of a pixel within the device context.
getSysBrush	Retrieves a handle to a logical brush that corresponds to the specified system color index.
getTextAlign	Gets the text alignment setting for the specified device context.

Method	Description
getTextExtent	Gets the bounding rectangle, as a *Size* object for the specified text, if it were to be drawn in the specified device context.
hScrollPos	Returns the position of the horizontal scroll bar in the dialog.
loadBitmap	Loads a bitmap from a file into memory and returns the handle to the bitmap.
objectToDC	Loads a graphic object into a device context.
opaqueText	Sets the text drawing mode in a device context to opaque, (background is redrawn with the current brush.)
rectangle	Draws a rectangle within the given device context.
removeBitmap	Frees an in-memory bitmap that was loaded through the *loadBitmap* method.
scroll	Scrolls the contents of the dialog's client area by the amount specified.
setArcDirection	Sets the current drawing direction.
setFont	Sets a new font to be used by the dialog.
setHScrollPos	Sets the thumb position of the horizontal scroll bar contained in the dialog.
setTextAlign	Sets the text alignment option for the specified device context.
setVScrollPos	Sets the thumb position of the vertical scroll bar contained in the dialog.
transparentText	Sets the text drawing mode in a device context to transparent, (background is not changed.)
vScrollPos	Returns the position of the vertical scroll bar in the dialog.
write	Writes text in a dialog using the given font, style, and color at the position specified.
writeDirect	Writes text in a dialog at the position specified using a device context.

4.11.2. createBrush

```
>>--createBrush(--+---------+--+-----------------+--)------------------------><
                  +--color--+  +-,-brushSpecifier-+
```

The *createBrush* method creates a logical brush that has the specified style, color, and pattern.

Arguments:

Both arguments are optional. If both arguments are omitted then a hollow brush is created. Otherwise, the arguments are:

color [optional]

The *color number*. When this argument is omitted and *brushSpecifier* is used, the color number defaults to 1.

brushSpecifier [optional]

If this argument is omitted a solid brush using the color specified is created. Otherwise, this argument can be the name of a bitmap file, or one of the following keywords. The keywords create a hatched brush. A bitmap file name will cause the bitmap to be loaded into memory and then used for the brush. Case is not significant in the keywords.

UPDIAGONAL	DOWNDIAGONAL
CROSS	HORIZONTAL
DIAGCROSS	VERTICAL

UPDIAGONAL
A 45-degree upward, left-to-right hatch.

CROSS
A horizontal and vertical crosshatch.

DIAGCROSS
A 45-degree crosshatch.

DOWNDIAGONAL
A 45-degree downward, left-to-right hatch.

HORIZONTAL
A horizontal hatch.

VERTICAL
A vertical hatch.

Remarks:

A brush is a bitmap that the operating system uses to paint the interiors of filled shapes. After the programmer creates a brush, it can then be selected into a device context using the *objectToDC*() method. When the brush is no longer needed use the *deleteObject*() method to release the operating resources used by the brush.

The *createBrush* method here is almost identical to the *createBrush* method of the *DialogExtensions* class. The method documented here is a method of the *WindowsExtensions* class and is therefore inherited by both the *dialog* and the dialog *control* objects.

However, in the dialog object, the *createBrush* method of the **`DialogExtensions`** class overrides this method. Therefore, this documentation is essentially the dialog control's *createBrush* documentation and the *createBrush* documentation is for the dialog object's *createBrush* method.

Details:

Sets the *.systemErrorCode*.

4.11.3. createFont

```
>>--createFont(--+----------+-+------------+-+---------+-+-------------+--)----><
                 +-fontName-+ +-,-fontSize-+ +-,-style-+ +-,-fontWidth-+
```

The createFont method creates a font and returns its handle.

Note the implementation of this method is incorrect. It does not return the font specified. The correct method to use is *createFontEx*(). **`createFont`**() is maintained for program compatibility. Some existing programs may rely on using the (incorrect) font returned.

Arguments:

The arguments are:

fontName [optional]
The name of the font. the default is System.

fontSize [optional]
The size of the font. The default is 10.

style [optional]

One or more of the following keyword, separated by blanks.

THIN	EXTRALIGHT	LIGHT
MEDIUM	SEMIBOLD	EXTRABOLD
BOLD	HEAVY	UNDERLINE
ITALIC	STRIKEOUT	

fontWidth [optional]

The average width of the characters in the font. If omitted, the width is set to the value of *fontSize*.

4.11.4. createFontEx

```
>>--createFontEx(--fontName-+-------------+--+--------------+--)------------><
                            +-,-pointSize--+  +-,-additional--+
```

Creates a logical font with the characteristics requested. This should be the preferred method to create fonts in ooDialog. The *createFont* method does **not** properly create the font requested.

Arguments:

The arguments are:

fontName [required]

The typeface name of the requested font. The *fonts* folder, accessed from the Control Panel on Windows, lists the typeface names of all fonts installed on the system.

As noted in the remarks section, the font mapper will try to substitute a similar font if the typeface name does not match any of the installed fonts.

pointSize [optional]

The point size of the requested font. The default is 8.

additional [optional]

Specifies additional characteristics of the requested font. If this argument is omitted, default values are used for all characteristics. The *additional* argument can be either:

- A **`.Directory`** object whose indexes specify the characteristics.
- A list of blank separated keywords that specify the characteristics.

Directory Object:

The indexes of the object specify the additional font characteristics. For any missing index, the default value is used for that characteristic. Unrecognized indexes are ignored. The indexes are listed below. Each index maps to an argument of the **`CreateFont()`** Win32 API. The Rexx programmer can, and maybe should, consult the MSDN *documentation* to get complete information on the individual arguments. The value for each index must be either numeric or logical, depending on the index.

The indexes below are listed in the same order as the arguments to **`CreateFont()`** from *nWidth* to *fdwPitchAndFamily*. Where an index has an internal link, the link provides some more information on the possible values for index. The valid indexes are:

WIDTH
: A non-negative whole number that specifies the average width of the characters in the requested font. The default is 0 which tells the font mapper to choose a closest match value.

ESCARPMENT
: A non-negative whole number that specifies the angle between the escapement vector and the x-axis of the device. The default is 0.

ORIENTATION
: A non-negative whole number that specifies the angle between each character's base line and the x-axis of the device. The default is 0.

WEIGHT
: A non-negative whole number in the range of 0 through 1,000 that specifies the weight of the font. The default is FW_NORMAL, which is 400.

ITALIC
: True for an italic font. The default is false.

UNDERLINE
: True for an underline font. The default is false.

STRIKEOUT
: True for a strikeout font. The default is false.

CHARSET
: A non-negative whole number that specifies the character set. The default is DEFAULT_CHARSET.

OUTPUTPRECISION
: A non-negative whole number that specifies the output precision. The default is OUT_TT_PRECIS.

CLIPPRECISION
: A non-negative whole number that specifies the clipping precision. The default is CLIP_DEFAULT_PRECIS.

QUALITY
: A non-negative whole number that specifies the output quality. The default is DEFAULT_QUALITY.

PITCHANDFAMILY
: A non-negative whole number that specifies the pitch and family of the font. The default is FF_DONTCARE.

Keyword List:

One or more of the following keywords, separated by blanks. Case is not significant. The UNDERLINE, ITALIC, and STRIKEOUT keywords specify the corresponding type of font. All the other keywords specify the weight of the font. Using more that one of the weight keywords is the list is undefined:

THIN	EXTRALIGHT	LIGHT
MEDIUM	SEMIBOLD	EXTRABOLD
BOLD	HEAVY	UNDERLINE
ITALIC	STRIKEOUT	

Return value:

The return is a handle to a font that can be used in any ooDialog method that requires a font handle. Because of the way the font mapper works, it is hard to conceive of a circumstance where this method would fail. However, if it were to fail, the *.systemErrorCode* would be non-zero.

Remarks:

The returned font can be used in any method that takes a font, such as, *setFont*, *setControlFont*, or *fontToDC*. When the programmer is done with the font, the operating system resources used by the font can be released with the *deleteFont* method.

The Windows operating system uses a font mapper to map a logical font to the characteristics of the font requested. Because available fonts can vary from system to system, the returned font is not always the same as the requested font. For example, if a font named Palatino is requested, but no such font is available on the system, the font mapper will substitute a font that has similar attributes but a different name.

The interface to the font mapper takes a wide range of arguments to specify many different attributes of the requested font. The *additional* argument gives the ooRexx programmer the ability to use any, or all, of these arguments. On the other hand, The *createFontEx* method has good defaults for each of the arguments, allowing the programmer to use the method with a minimum of effort.

If an application will run on different systems, where the programmer does not know in advance what fonts will be available on each system, carefully specifying the font characteristics will help give the application the same look on each system. The font mapper will pick a font on each system that most closely matches the requested attributes.

Details:

This method provides an interface to the Win32 API: **`CreateFont()`**. Use the MSDN *documentation* documentation to get more information on the arguments to this method.

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect arguments are detected.

Example:

This example creates a 16 point, italic, underlined, Ariel font and then uses the font for a static control.

```
additional = .directory~new
additional~italic = .true
additional~underline = .true

hFont = self~createFontEx("Arial", 16, additional)
hOldFont = self~newStatic(IDC_ST_ALERT)~setFont(hFont)
...
```

This example creates a 14 point, bold, italic, Times New Roman font and instructs the font mapper to only choose from true type fonts. If there are no true type fonts installed on the system, then the font mapper returns to its default behavior.

```
FW_BOLD = 700
OUT_TT_ONLY_PRECIS = 7

additional = .directory~new
additional~italic = .true
additional~weight = FW_BOLD
```

```
  additional~outputPrecision = OUT_TT_ONLY_PRECIS

  hFont = self~createFontEx("Times New Roman", 14, additional)
```

4.11.4.1. createFontEx Argument Values

The following list is the indexes of the **`.Directory`** object that can be used for the third argument of the *createFontEx*() method. The list provides additional information on the meaning of values that can be assigned to each index.

width Must be an integer.

Specifies the average width, in logical units, of the characters in the requested font. The default value of zero, tells the font mapper to choose a closest match value. This is likely to produce the best results, unless the programmer has some need that requires specifying the character width.

The closest match value is determined by comparing the absolute values of the difference between the current device's aspect ratio and the digitized aspect ratio of available fonts.

escarpment Must be an integer.

Specifies the angle, in tenths of degrees, between the escapement vector and the x-axis of the device. The escapement vector is parallel to the base line of a row of text. Windows has no predefined values for this argument.

orientation Must be an integer.

Specifies the angle, in tenths of degrees, between each character's base line and the x-axis of the device. Windows has no predefined values for this argument.

weight Must be an integer.

Specifies the weight of the font in the range 0 through 1000. For example, 400 is normal and 700 is bold. If this value is zero, a default weight is used. The following table shows Windows predefined values for this argument:

Value	Weight
FW_DONTCARE	0
FW_THIN	100
FW_EXTRALIGHT	200
FW_ULTRALIGHT	200
FW_LIGHT	300
FW_NORMAL	400
FW_REGULAR	400
FW_MEDIUM	500
FW_SEMIBOLD	600
FW_DEMIBOLD	600
FW_BOLD	700
FW_EXTRABOLD	800
FW_ULTRABOLD	800
FW_HEAVY	900
FW_BLACK	900

italic Must be a logical.

Set this index to `.true` to request an italic font. The default is `.false`.

underline Must be a logical.

Set this index to `.true` to request an underlined font. The default is `.false`.

strikeout Must be a logical.

Set this index to `.true` to request a strike out font. The default is `.false`.

charset Must be an integer.

Specifies the character set. The following values are predefined in Windows:

Symbol	Value
ANSI_CHARSET	0
BALTIC_CHARSET	186
CHINESEBIG5_CHARSET	136
DEFAULT_CHARSET	1
EASTEUROPE_CHARSET	238
GB2312_CHARSET	134
GREEK_CHARSET	161
HANGUL_CHARSET	129
MAC_CHARSET	77
OEM_CHARSET	255
RUSSIAN_CHARSET	204
SHIFTJIS_CHARSET	128
SYMBOL_CHARSET	2
TURKISH_CHARSET	162
VIETNAMESE_CHARSET	163

outputPrecision Must be an integer.

The output precision defines how closely the output must match the requested font's height, width, character orientation, escapement, pitch, and font type.

Applications can use the OUT_DEVICE_PRECIS, OUT_RASTER_PRECIS, OUT_TT_PRECIS, and OUT_PS_ONLY_PRECIS values to control how the font mapper chooses a font when the operating system contains more than one font with a specified name. For example, if an operating system contains a font named Symbol in raster and TrueType form, specifying OUT_TT_PRECIS forces the font mapper to choose the TrueType version. Specifying OUT_TT_ONLY_PRECIS forces the font mapper to choose a TrueType font, even if it must substitute a TrueType font of another name.

It can be one of the following values:

Symbol	Meaning	Value
OUT_CHARACTER_PRECIS	Not used.	2
OUT_DEFAULT_PRECIS	Specifies the default font mapper behavior.	0

Symbol	Meaning	Value
OUT_DEVICE_PRECIS	Instructs the font mapper to choose a Device font when the system contains multiple fonts with the same name.	5
OUT_OUTLINE_PRECIS	This value instructs the font mapper to choose from TrueType and other outline-based fonts.	8
OUT_PS_ONLY_PRECIS	Instructs the font mapper to choose from only PostScript fonts. If there are no PostScript fonts installed in the system, the font mapper returns to default behavior.	10
OUT_RASTER_PRECIS	Instructs the font mapper to choose a raster font when the system contains multiple fonts with the same name.	6
OUT_STRING_PRECIS	Not used.	1
OUT_STROKE_PRECIS	Not used.	3
OUT_TT_ONLY_PRECIS	Instructs the font mapper to choose from only TrueType fonts. If there are no TrueType fonts installed in the system, the font mapper returns to default behavior.	7
OUT_TT_PRECIS	Instructs the font mapper to choose a TrueType font when the system contains multiple fonts with the same name.	4

clipPrecision Must be an integer.

The clipping precision defines how to clip characters that are partially outside the clipping region. It can be one or more of the following values. Use .DlgUtil~*or* to combine values.

Symbol	Meaning	Value
CLIP_CHARACTER_PRECIS	Not used.	1 (0x01)
CLIP_DEFAULT_PRECIS	Specifies default clipping behavior.	0 (0x00)
CLIP_DFA_DISABLE	Windows XP SP1: Turns off font association for the font. Note that this flag is not guaranteed to have any effect on any platform after Windows Server 2003.	64 (0x40)
CLIP_EMBEDDED	You must specify this flag to use an embedded read-only font.	128 (0x80)
CLIP_LH_ANGLES	When this value is used, the rotation for all fonts depends on whether the orientation of the coordinate system is left-handed or right-handed. If not used, device fonts always rotate counterclockwise, but the rotation of other fonts is dependent on the orientation of the coordinate system. For more information about the orientation of coordinate systems, see the description of the orientation parameter.	16 (0x10)
CLIP_MASK	Not used.	15 (0x0f)
CLIP_STROKE_PRECIS	Not used.	2 (0x02)
CLIP_TT_ALWAYS	Not used.	32 (0x20)

quality Must be an integer.

The output quality defines how carefully the font mapper must attempt to match the logical-font attributes to those of an actual physical font.

Note that if neither ANTIALIASED_QUALITY nor NONANTIALIASED_QUALITY is selected, the font is antialiased only if the user chooses "smooth screen fonts" in Control Panel. Quality can be one of the following values:

Symbol	Meaning	Value
ANTIALIASED_QUALITY	Font is antialiased, or smoothed, if the font supports it and the size of the font is not too small or too large.	4
CLEARTYPE_QUALITY	If set, text is rendered (when possible) using ClearType antialiasing method. See Remarks for more information.	6
DEFAULT_QUALITY	Appearance of the font does not matter.	0
DRAFT_QUALITY	Appearance of the font is less important than when the PROOF_QUALITY value is used. For GDI raster fonts, scaling is enabled, which means that more font sizes are available, but the quality may be lower. Bold, italic, underline, and strikeout fonts are synthesized, if necessary.	1
NONANTIALIASED_QUALITY	Font is never antialiased, that is, font smoothing is not done.	3
PROOF_QUALITY	Character quality of the font is more important than exact matching of the logical-font attributes. For GDI raster fonts, scaling is disabled and the font closest in size is chosen. Although the chosen font size may not be mapped exactly when PROOF_QUALITY is used, the quality of the font is high and there is no distortion of appearance. Bold, italic, underline, and strikeout fonts are synthesized, if necessary.	2

pitchAndFamily Must be an integer.

Specifies the pitch and family of the font. Font families describe the look of a font in a general way. They are intended for specifying fonts when the exact typeface requested is not available.

Combine one pitch value with one family value. The values are combined using a boolean or, but in this case they could simply be added. The programmer can also use .DlgUtil~*or* to combine values.

Symbol	Meaning	Value
Pitch		
DEFAULT_PITCH		0
FIXED_PITCH		1
VARIABLE_PITCH		2
Family		
FF_DECORATIVE	Novelty fonts. Old English is an example.	128 (0x80)
FF_DONTCARE	Use default font.	4 (0x04)

Symbol	Meaning	Value
FF_MODERN	Fonts with constant stroke width, with or without serifs. Pica, Elite, and Courier New are examples.	32 (0x20)
FF_ROMAN	Fonts with variable stroke width and with serifs. MS Serif is an example.	8 (10x08)
FF_SCRIPT	Fonts designed to look like handwriting. Script and Cursive are examples.	64 (0x40)
FF_SWISS	Fonts with variable stroke width and without serifs. MS Sans Serif is an example.	16 (0x10)

4.11.5. createPen

```
>>--createPen(--+-------+--+----------+--+----------+--)----------------------><
                +-width-+  +-,-style--+  +-,-color--+
```

Creates a logical pen that has the specified style, width, and color.

Arguments:

The arguments are:

width [optional]

A non-negative whole number that specifies the width of the lines that the pen draws. A width of 1 is the default. If 0 is specified the operating system will change the width to 1.

style [optional]

One of the following keywords, case is not significant. Solid is the default.

SOLID	DOT	DASHDOTDOT
DASH	DASHDOT	NULL

SOLID

The line drawn by the pen is a solid line. This line can be any thickness. SOLID is the default.

DASH

The pen will draw a dashed line. The width must be 1.

DOT

The pen will draw a dotted line. The width must be 1.

DASHDOT

The pen will draw a line consisting of a dash followed by a dot. The width must be 1.

DASHDOTDOT

The pen will draw a line consisting of a dash followed by two dots. The width must be 1.

DASHDOTDOT

The pen will draw an invisible line. The width can be any thickness.

color [optional]

The *color number* for the pen. 0 is the default.

Remarks:

After a pen is selected into a device context, it can be used to draw lines and curves. It can then be selected into a device context using the *objectToDC*() method. When the pen is no longer needed use the *deleteObject*() method to release the operating resources used by the brush.

For dashed and dotted lines, if the width specified is greater than 1, the operating system will return a pen of that width, but change its style to solid.

Details:

Sets the *.systemErrorCode*.

Example:

The following example creates a dotted pen object with a width of 1:

```
hPen = dlg~createPen(1, "DOT", 13)
```

4.11.6. deleteFont

```
>>--deleteFont(--hFont--)---------------------------><
```

The deleteFont method deletes a font. This method is to be used to delete a font created with the *createFontEx*() method.

Arguments:

The only argument is:

hFont

The handle of a font.

4.11.7. deleteObject

```
>>--deleteObject(--obj--)---------------------------><
```

The deleteObject method deletes a graphic object, namely a pen or a brush. See *createPen* and *createBrush* for information on how to get the handle of a pen or brush.

Arguments:

The only argument is:

obj

The handle of a pen or brush.

4.11.8. drawAngleArc

```
>>--drawAngleArc(-dc-,-xs-,-ys-,-x-,-y-,-radius-,-startangle-,-sweepangle-)----><
```

The drawAngleArc method draws a partial circle (arc) and a line connecting the start drawing point with the start of the arc on the given device context using the active pen for the outline. The circle is drawn counterclockwise with the given radius between the given angles.

Arguments:

The arguments are:

dc

The device context.

xs, ys

The start draw position, in pixels.

x, y

The center of the circle, in pixels.

radius

The radius of the circle, in pixels.

startangle, sweepangle

The starting and ending angles for the partial circle in degrees (0 is the x-axis).

4.11.9. drawArc

```
>>--drawArc(--dc-,-x-,-y-,-x2-,-y2--+-------+--+-------+--+-------+--+-------+--)---><
                                    +-,-r1x-+  +-,-r1y-+  +-,-r2x-+  +-,-r2y-+
```

The drawArc method draws a circle or ellipse on the given device context using the active pen for the outline. The circle or ellipse is drawn within the boundaries of an imaginary rectangle whose coordinates are given. A partial figure can be drawn by giving the end points of two radials. By default, the figure is drawn counterclockwise, but the direction can be modified using *setArcDirection*.

Arguments:

The arguments are:

dc

The device context.

x, y

The position of the upper left corner of the imaginary rectangle, in pixels.

x2, y2

The position of the lower right corner of the imaginary rectangle, in pixels.

r1x, r1y, r2x, r2y

The end points of the starting and ending radials for drawing the figure. A full circle or ellipse is drawn if no start and end are given. Omitted values default to 0. Imaginary radials are drawn from the center of the bounding rectangle to the start and end points. The circle or ellipse is then drawn between the intersections of these lines with the full circle or ellipse.

Example:

This example draws a full ellipse and a quarter circle:

```
dc = self~getControlDC(100)
pen = self~createPen(4,"solid",13)
oldp = self~objectToDC(dc,pen)
self~drawArc(dc,50,50,200,150)                  /* full ellipse */
self~drawArc(dc,100,100,150,150, 200,50,75,75) /* quarter circle */
self~objectToDC(dc,oldp); self~deleteObject(pen)
```

4.11.10. drawLine

```
>>--drawLine(--dc--,--+-------+--,--+-------+--,--toX--,--toY--)-><
                      +-fromX-+     +-fromY-+
```

The drawLine method draws a line within the device context using the active pen.

Arguments:

The arguments are:

dc
: The device context.

fromX, fromY
: The starting position, in pixels. If omitted, the previous end point of a line or arc is used.

toX, toY
: The target position.

4.11.11. drawPie

```
>>--drawPie(--dc-,-x-,-y-,-x2-,-y2--+-------+--+-------+--+-------+--+-------+--)---><
                                    +-,-r1x-+  +-,-r1y-+  +-,-r2x-+  +-,-r2y-+
```

The drawPie method draws a pie of a circle or ellipse on the given device context using the active pen for the outline and the active brush to fill the pie. The circle or ellipse is drawn within the boundaries of an imaginary rectangle whose coordinates are given. The arc is drawn between start and end radials in the direction specified by *setArcDirection*.

Arguments:

The arguments are:

dc
: The device context.

x, y
: The position of the upper left corner of the imaginary rectangle, in pixels.

x2, y2
: The position of the lower right corner of the imaginary rectangle.

r1x, r1y, r2x, r2y
: The end points of the two radials (same as for *drawArc*).

4.11.12. drawPixel

```
>>--drawPixel(--dc--,--x--,--y--,--color--)----------><
```

The drawPixel method draws a pixel within the device context.

Arguments:

The arguments are:

dc
: The device context.

x, y
: The position, in pixels.

color
: The *color number* for the pixel.

4.11.13. fillDrawing

```
>>--fillDrawing(--dc--,--x--,--y--,--color--)---><
```

Fills an area of the display surface with the current brush. The area to be filled is outlined by the color specified.

Arguments:

The arguments are:

dc [required]
: The device context containing the area to be filled.

x [required]
: The x coordinate of the position in the area to start filling.

y [required]
: The x coordinate of the position in the area to start filling.

color [required]
: Specifies the color that outlines the area to be filled. The color can be specified as a color palette *index*, or as a *COLORREF* color.

Return value:

Returns true on success, false on error.

Remarks:

The operating system assumes that the area to be filled is completely bounded by the color specified by the *color* argument. The operating system begins filling at the point specified by the *x* and *y* arguments and continues in all directions until it reaches the boundary.

The MSDN documentation says there are a number of reasons why this method may fail, and lists some of the reasons as being:

- The filling could not be completed.
- The specified point has the boundary color specified by the *color* argument.
- The point is outside the clipping region, that is, it is not visible on the device.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable. If *dc* is zero, the error code is set to: 1, *ERROR_INVALID_FUNCTION Incorrect function.* The operating system may set other error codes for other reasons.

4.11.14. fillRect

```
>>--fillRect(--dc--,--rect--,--brush--)----------><
```

Fills a rectangle using the specified brush within the specified device context.

Arguments:

The arguments are:

dc [required]

The device context that contains the rectangle.

rect [required]

A *Rect* object that specifies the rectangle to be filled.

brush [required]

Specifies the brush that is to be used to fill the rectangle. This can either be the handle of the brush or the index of the system color to be used. Brush handles are retrieved using methods such as *createBrush* or *getSysBrush*. A system color index can be either the non-negative whole number ID or the keyword ID. Both types of IDs can be looked up in the System Color Elements *table*.

Return value:

Returns true on success, false on error.

Remarks:

Unlike the *rectangle* method, the *fileRect* method does not outline the rectangle with the pen of the device context.

In addition, for programmers that may read the Microsoft *documentation* on the Windows *FillRect* API, the *fillRect* method fills the entire rectangle specified. The Windows API does not include the rectangle's right and bottom sides. That API fills a rectangle up to, but not including, the right column and bottom row. If the ooDialog programmer needs that behaviour, she should subtract 1 from the right and bottom coordinates of the *rect* object.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable. If *dc* is zero, the error code is set to: 1, *ERROR_INVALID_FUNCTION Incorrect function.* The operating system may set other error codes for other reasons.

Example:

This example comes from a clock program that displays the digital time. The time is updated every second. If the previous time is not erased, it just becomes a smear as the different digits are written over the top of each other. The example gets a system brush that is the color of the background of the clock and uses that to erase the background of the area that displays the digital time before it is updated with the current time:

```
  -- Get a system brush using the button's system color. Do not delete system
  -- brushes
  btnBrush = self~getSysBrush('BTNFACE')
  curBrush = self~objectToDC(dc, btnBrush)

  -- Erase the area where the time will be displayed by painting the
```

```
-- background with our button brush.  Get the size of the area, the
-- text extent, taken up by the time.  Use that to calculate our
-- rectangle.
s = button~getTextExtent(dc, time)
r = .Rect~new(290, 40, 290 + s~width, 40 + s~height)
self~fillRect(dc, r, btnBrush)
```

4.11.15. fontColor

```
>>--fontColor(--color--,--dc--)----------------------><
```

The fontColor method sets the font color for a device context.

Arguments:

The arguments are:

color

The index of a color in the system's color palette.

dc

The device context.

4.11.16. fontToDC

```
>>--fontToDC(--dc--,--hFont--)-----------------------><
```

The fontToDC method loads a font into a device context and returns the handle of the previous font. Use the *getWindowDC*, *getDC*, or *getControlDC* method to retrieve a device context, and the *createFontEx*() method to get a font handle. To reset the font to the original state, use another fontToDC call with the handle of the previous font. To release the device context, use the *freeWindowDC*, *freeDC*, or *freeControlDC* method.

Arguments:

The arguments are:

dc

The device context of a dialog or button.

hFont

The handle of a font.

Example:

This example loads an Arial font into the current dialog window:

```
additional = .directory~new
additional~italic = .true
hfnt = MyDialog~createFontEx("Arial", 16, additional)
dc   = MyDialog~getDC
oldf = MyDialog~fontToDC(dc,hfnt)  /* activate font */
...
MyDialog~fontToDC(dc,oldf)         /* restore previous font */
MyDialog~freeDC(dc)
```

4.11.17. freeDC

```
>>--freeDC(--dc--)-------------------------------><
```

The *freeDC* method releases the device context resources that were reserved by the *getDC* method.

Arguments:

The single argument is:

dc [required]

The *handle* to the device context that is to be released.

Return value:

Returns 0 on success, 1 on error.

Remarks:

The programmer should always free a device context when the application is done using the device context.

Example:

This example gets the device context of a button window, draws something in the client space of the button, and then releases the device context:

```
pb = self~newPushButton("DRAWINGS")
if pb = .nil then return -1
dc = pb~getDC
if dc = 0 then return -1
... /* draw something */
pb~freeDC(dc)
```

4.11.18. getArcDirection

```
>>--getArcDirection(--dc--)--------------------------><
```

The getArcDirection method returns the current drawing direction for the *drawArc* method.

Arguments:

The only argument is:

dc

The device context.

4.11.19. getDC

```
>>--getDC--------------------------------------------><
```

The getDC method reserves drawing resources and returns the handle to the display device context of a dialog or dialog control.

Return value:

The handle to the device context, or 0 if this method failed.

Example:

The following example retrieves the device context of button DRAWINGS, processes the drawing commands, and frees the device context resources:

```
obj = MyDialog~newPushButton("DRAWINGS")
if obj = .Nil then return -1
dc = obj~getDC
if dc = 0 then return -1
... /* draw something */
obj~freeDC(dc)
```

Note

When you have finished with the device context, call *freeDC*.

4.11.20. getFont

```
>>--getFont------------------------------------------><
```

Retrieves a handle to the font currently being used by the dialog or dialog control.

Arguments:

There are no arguments for this method

Return value:

This method returns a handle to the font for text used by the dialog or dialog control.

Example:

This is an example from a fictious program where a custom font is used to emphasis the text in two static controls. When the user does some action, the text in one control is emphasised and the other is de-emphasised. One way to do this is to just swap the fonts.

```
::method onToggle private
  static1 = self~newStatic(IDC_ST_OUTSTANDING_BALANCE)
  static2 = self~newStatic(IDC_ST_LASTPAYMENT)

  font1 = static1~getFont
  font2 = static2~getFont
  static1~setFont(font2)
  static2~setFont(font1)
```

4.11.21. getPixel

```
>>--getPixel(--dc--,--x--,--y--)---------------------><
```

The getPixel method returns the color number of a pixel within the device context.

Arguments:

The arguments are:

dc
: The device context.

x, y
: The position, in pixels.

4.11.22. getSysBrush

```
>>--getSysBrush(--sysColor--)--------------------><
```

Retrieves a handle to a logical brush that corresponds to the specified system color index.

Arguments:

The single argument is:

sysColor [required]
: Specifies the system color to be used. This can be either the non-negative whole number ID or the keyword ID. Both types of IDs can be looked up in the System Color Elements *table*.

Return value:

The handle to the brush on success, or .nil on failure.

Remarks:

A brush is a bitmap that the system uses to paint the interiors of filled shapes. System color brushes track changes in system colors. In other words, when the user changes a system color, the associated system color brush automatically changes to the new color.

To paint with a system color brush, an application should use *getSysBrush* instead of using *createBrush*, because *getSysBrush* returns a cached brush instead of allocating a new one. System color brushes are owned by the system and must not be destroyed.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example gets a brush with the system color of a push button and then selects it into a device context to use it form painting:

```
  -- Get a system brush using the button's system color. Do not delete system
  -- brushes
  btnBrush = self~getSysBrush('BTNFACE')
  curBrush = self~objectToDC(dc, btnBrush)
```

4.11.23. getTextAlign

```
>>--getTextAlign(--hDC--)------------------------><
```

Gets the text alignment setting for the specified device context.

Arguments:

The single argument is:

hDC [required]
: The device context whose text alignment is required.

Return value:

A string of keywords specifying the text alignment. The returned string will contain exactly one keyword from the following 3 groups of keywords, in the same order as the groups are listed:

LEFT
: The reference point is on the left edge of the bounding rectangle.

RIGHT
: The reference point is on the right edge of the bounding rectangle.

CENTER
: The reference point is aligned horizontally with the center of the bounding rectangle.

TOP
: The reference point is on the top edge of the bounding rectangle.

BOTTOM
: The reference point is on the bottom edge of the bounding rectangle.

BASELINE
: The reference point is on the base line of the text.

NOUPDATECP
: The current position is not updated after each text output call.

UPDATECP
: The current position is updated after each text output call.

Remarks:

The bounding rectangle is a rectangle bounding all of the character cells in a string of text. Its dimensions can be obtained by using the *getTextExtent* method.

The text-alignment flags determine how the text writing methods, such as *write* and *writeDirect*, align a string of text in relation to the string's reference point provided to the methods.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

4.11.24. getTextExtent

```
>>--getTextExtent(--hDC--,--text--)--------------><
```

Gets the bounding rectangle, as a *Size* object for the specified text, if it were to be drawn in the specified device context.

Arguments:

The arguments are:

hDC [required]
: The device context in which the text will be drawn.

text
: The text string to be drawn in the device context.

Return value:

A `Size` object that reflects the size of the bounding rectangle of the text.

Remarks:

The size is calculated using the currently selected font in the device context.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

4.11.25. hScrollPos

```
>>--hScrollPos---------------------------------------><
```

The HScrollPos method returns the position of the horizontal scroll bar in the associated dialog or dialog control.

Return value:

The position of the horizontal scroll bar.

4.11.26. loadBitmap

```
>>--loadBitmap(--bmpFilename--+-----------+--)------><
                              +-,-loadOpt--+
```

The *loadBitmap* method loads a bitmap from a file into memory and returns the handle to the bitmap.

In general, the *Image* class should be used when working with bitmaps, if possible. The *loadBitmap*() and *removeBitmap*() methods date back to the Windows 3.1 ooDialog and use outdated techniques. However, at this time, the bitmap button methods, like *installBitmapbutton*() still require the bitmap handle returned from *loadBitmap*() method.

Arguments:

The arguments are:

bmpFilename [required]
: The file name of the bitmap file. The name can be a relative or absolute path.

loadOpt [optional]
: The only load option is: USEPAL. This sets the color palette of the bitmap as the system color palette.

Return value:

Returns the handle of the loaded bitmap on success, or 0 on error.

Remarks:

Use the *removeBitmap*() method to free memory when the bitmap is no longer in use. Although it does no harm, it is not necessary to free the bitmap if the Rexx program is ending. When the Rexx interpreter process ends, the operating system will free the bitmap memory. To use the bitmap in the *installBitmapbutton*() or *changeBitmapButton*() methods specify the INMEMORY option.

Details:

Sets the *.systemErrorCode*.

Example:

The following example loads the bitmap file, **`Walker.bmp`**, into memory. The file is located in the **`bmp`** subdirectory. hBmp is the handle to this in-memory bitmap.

```
   hBmp = MyDialog~loadBitmap("bmp\Walker.bmp", "USEPAL")
```

4.11.27. objectToDC

```
>>--objectToDC(--dc--,--obj--)-----------------------><
```

The objectToDC method loads a graphic object, namely a pen or a brush, into a device context. Subsequent lines, rectangles, and arcs are drawn using the pen and brush.

Arguments:

The arguments are:

dc

The device context.

obj

The object: a pen or a brush.

Return value:

The handle of the previous active pen or brush. It can be used to restore the previous environment.

Example:

The following example activates a pen for drawing:

```
dc = MyBaseDialog~getDC
hpen = MyDialog~createPen(2, "SOLID", 4)
MyDialog~objectToDC(dc,hpen)
... /* do lines, rectangles, ... */
MyDialog~deleteObject(hpen)
```

4.11.28. opaqueText

```
>>--opaqueText(--dc--)-------------------------------><
```

The opaqueText method restores the default text mode, that is, with a white background behind the text, which overlays whatever is at that position in the dialog or dialog control. Use this method after transparent mode was set using *transparentText*.

Arguments:

The only argument is:

dc

A device context.

4.11.29. rectangle

```
>>--rectangle(--dc--,--x--,--y--,--x2--,--y2--+-----------+--)-----------------><
                                              +-,-keyWord-+
```

The rectangle method draws a rectangle to the given device context. The appearance is determined by the graphics objects currently active in the device context. The active pen draws the outline and, optionally, the active brush fills the inside area. The default pen is thin black and the default brush is white.

Arguments:

The arguments are:

dc

The device context.

x, y

The position of the upper left corner of the rectangle, in pixels.

x2, y2

The position of the lower right corner.

keyWord [optional]

If the argument is omitted, the rectangle is just outlined (drawn) with the active pen. otherwise the rectangle is filled with the active brush. The keyword would be "FILL" to file the rectangle, but the argument is not actually checked. If the argument is used, the rectangle is filled, if it is not used the rectangle is outlined..

Example:

The following example draws a red rectangle filled with yellow, surrounded by a black rectangle:

```
dc = self~getControlDC(100)
brush = self~createBrush(15)        /* yellow */
pen = self~createPen(10,"solid",13) /* thick red */
oldb = self~objectToDC(dc,brush)
oldp = self~objectToDC(dc,pen)
self~rectangle(dc, 50, 50, 200, 150, "FILL")
self~objectToDC(dc,oldp); self~deleteObject(pen)
self~objectToDC(dc,oldb); self~deleteObject(brush)
self~rectangle(dc, 40, 40, 210, 160) /* default */
```

4.11.30. removeBitmap

```
>>--removeBitmap(--hBitmap--)------------------------><
```

Use this method to free an in-memory bitmap that was loaded through the *loadBitmap*() method.

Arguments:

The only argument is:

hBitmap [required]
: The bitmap handle.

Return value:

Returns 0 on success or 1 of the bitmap handle is not valid.

Remarks:

This method frees the memory used by the bitmap. Do not free a bitmap that is in use. It is not necessary to free the bitmap if the Rexx program is ending. When the Rexx interpreter process ends, the operating system will free the bitmap memory. However, in a long running program, the programmer will likely want to free bitmaps that are not in use.

4.11.31. scroll

```
>>--scroll(--cx--,--cy--)---------------------------><
```

The Scroll method scrolls the contents of the associated dialog or dialog control by the amount specified.

Arguments:

The arguments are:

cx
: The number of screen pixels the content of the dialog or dialog control is to be scrolled to the right or to the left, if negative.

cy
: The number of screen pixels the content of the dialog or dialog control is to be scrolled downward or upward, if negative.

Return value:

0
: Scrolling was successful.

1
: Scrolling failed.

4.11.32. setArcDirection

```
>>--setArcDirection(--dc--+--------------+--)--------><
                          +-,-direction--+
```

The setArcDirection method changes the drawing direction for the *drawArc* and *drawPie* methods. The default direction is counterclockwise.

Arguments:

The arguments are:

dc [required]
: The device context.

direction [optional]

The new drawing direction. Use either the CLOCKWISE or COUNTERCLOCKWISE keywords, case is not significant. If the argument is omitted, then the direction is set back to the default, counterclockwise.

4.11.33. setFont

```
>>--setFont(--fontHandle--+----------+--)-----------><
                          +-,-redraw-+
```

The setFont method assigns another font to be used for the text in a dialog or dialog control.

Arguments:

The arguments are:

fontHandle

The handle to the font that is to be used by the dialog or dialog control. There are several methods to get the font handle, including *createFontEx*() or *getFont*().

redraw

Optional, .true or .false. If you specify .true, the message sent to the underlying dialog or dialog control tells it to redraw itself. If you specify .false, the dialog or dialog control is not told to redraw itself. The default is .true.

Return value:

This method always returns 0.

Example:

The following example creates the font Arial with a size of 14 and assigns it to the tree view control FILES, which is forced to be redrawn.

```
hfnt = dlg~createFontEx("Arial", 14)
dlg~newTreeView("FILES")~setFont(hfnt, .true)
```

4.11.34. setHScrollPos

```
>>--setHScrollPos(--position--+-----------+--)-------><
                              +-,-redraw--+
```

The SetHScrollPos method sets the thumb position of the horizontal scroll bar contained in the associated dialog or dialog control.

Arguments:

The arguments are:

position

The new thumb position of the horizontal scroll bar.

redraw

If this argument is 1 (the default), the display of the scroll bar is updated.

Return value:

The previous position of the horizontal scroll bar, or 0 if this method failed.

4.11.35. setTextAlign

```
>>--setTextAlign(--hDC--,--align--)--------------><
```

Sets the text alignment option for the specified device context.

Arguments:

The arguments are:

hDC [required]

The device context whose text alignment is to be set.

align [optional]

A string of keywords specifying the text alignment. The following keywords are recognized. Specify exactly one keyword from the following 3 groups of keywords, case and order are insignificant.

The default if this argument is omitted is: **`LEFT TOP NOUPDATECP`**. For any group that has no keyword specified, the default for that group is used. I.e., if neither LEFT RIGHT nor CENTER is specified, LEFT is used.

LEFT

The reference point is on the left edge of the bounding rectangle.

RIGHT

The reference point is on the right edge of the bounding rectangle.

CENTER

The reference point is aligned horizontally with the center of the bounding rectangle.

TOP

The reference point is on the top edge of the bounding rectangle.

BOTTOM

The reference point is on the bottom edge of the bounding rectangle.

BASELINE

The reference point is on the base line of the text.

NOUPDATECP

The current position is not updated after each text output call.

UPDATECP

The current position is updated after each text output call.

Return value:

Returns the previous alignment of the device context as a string of keywords. The string is identical to what the *getTextAlign* method would have returned if the method was invoked immediately prior to invoking this method. On error the empty string is returned.

Remarks:

The text-alignment flags determine how the text writing methods, such as *write* and *writeDirect*, align a string of text in relation to the string's reference point provided to the methods.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

4.11.36. setVScrollPos

```
>>--setVScrollPos(--position--+-----------+--)-------><
                              +-,-redraw--+
```

The SetVScrollPos method sets the thumb position of the vertical scroll bar contained in the associated dialog or dialog control.

Arguments:

The arguments are:

position

The new thumb position of the vertical scroll bar.

redraw

If this argument is 1 (the default), the display of the scroll bar is updated.

Return value:

The previous position of the vertical scroll bar, or 0 if this method failed.

4.11.37. transparentText

```
>>--transparentText(--dc--)-------------------------><
```

The transparentText method enables you to write text to a device context using *writeDirect* in transparent mode, that is, without a white background behind the text. Restore the default mode using *opaqueText*.

Arguments:

The only argument is:

dc

A device context.

4.11.38. vScrollPos

```
>>--vScrollPos---------------------------------------><
```

The VScrollPos method returns the position of the vertical scroll bar in the associated dialog or dialog control.

Return value:

The position of the vertical scroll bar.

4.11.39. write

```
>>--write(-x-,-y-,-text-+---------+-+---------+-+--------+-+------+-+------+-)---><
                         +-,-fName-+ +-,-fSize-+ +-,-opts-+ +-,-fg-+ +-,-bk-+
```

The *write* method writes the specified text to the dialog or dialog control in the given font, style, and color at the specified position.

Arguments:

The arguments are:

x, y [required]

The starting position of the text, in pixels. The position coordinates are relative to the window, or the client area of the window, not relative to the screen.

text [required]

The string to be written.

fName [optional]

The font name. The default if omitted is SYSTEM.

fSize [optional]

The point size of the font. If omitted, the standard size (10) is used.

opts [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. These options control aspects of the font and how the font is written.

OPAQUE	LIGHT	BOLD
TRANSPARENT	MEDIUM	UNDERLINE
CLIENT	SEMIBOLD	ITALIC
THIN	EXTRABOLD	STRIKEOUT
EXTRALIGHT	HEAVY	

OPAQUE

The background of the area the text will occupy is painted with the specified background color, or with white if the background color is omitted, before writing the text. This has the effect of "erasing" whatever is currently drawn in that area. Contrast this with the TRANSPARENT option. OPAQUE is the default.

TRANSPARENT

The background area of the text is left unchanged. (The background color option is ignored if it is used.) This has the effect of writing the text over the top of whatever is currently drawn in the area the text will occupy. Contrast this with the OPAQUE option.

CLIENT

The position for the text will be relative to the *client area* of the dialog or dialog control rather than relative to the window itself.

THIN

The weight of the font in a range of 0 through 1000 will be 100.

EXTRALIGHT

The weight of the font in a range of 0 through 1000 will be 200.

LIGHT

The weight of the font in a range of 0 through 1000 will be 300.

MEDIUM
: The weight of the font in a range of 0 through 1000 will be 500.

SEMIBOLD
: The weight of the font in a range of 0 through 1000 will be 600.

BOLD
: The weight of the font in a range of 0 through 1000 will be 700.

EXTRABOLD
: The weight of the font in a range of 0 through 1000 will be 800.

HEAVY
: The weight of the font in a range of 0 through 1000 will be 900.

UNDERLINE
: An underline font is used.

ITALIC
: An italic font is used.

STRIKEOUT
: A strike out font is used.

fg [optional]
: The color index for the text foreground color. If omitted, the text color is left unchanged.

bk [optional]
: The color index of the background color. If omitted, the background color is left unchanged. The background color is not used in transparent mode.

Return value:

Returns 0 on success and 1 on error.

Remarks:

This method sets **`.systemErrorCode`** to the error code set by the operating system when a failure in one of the Win32 APIs is detected. However, there is one Win32 API, **`SelectObject()`**, that does not set the system error code on failure. It is unlikely that it will fail, but if it does, the ooDialog framework sets **`.systemErrorCode`** to **`156`**, ERROR_SIGNAL_REFUSED.

The text message for error code **`156`** is: *The recipient process has refused the signal.* In this case, the text message is not really related to the failure, it is just used to indicate that the **`SelectObject()`** API failed.

Details:

Sets the *.systemErrorCode*. See the remarks above.

Example:

The following example writes the string "Hello world!" to the dialog using a blue 24pt Arial font in bold and transparent, italic style:

```
dlg~write(5, 5, "Hello world!", "Arial", 24, "BOLD ITALIC TRANSPARENT CLIENT", 4)
```

4.11.40. writeDirect

```
>>--writeDirect(--dc--,--xPos--,--yPos--,--text--)---><
```

The writeDirect method enables you to write text to a device context at a given position.

Arguments:

The arguments are:

dc
: A device context.

xPos, yPos
: The position where the text is placed, in pixels.

text
: The string you want to write to the dialog or dialog control.

Property Sheet and Control Dialogs

ooDialog supports two approaches to embedding content in a *Tab* control. Although the appearance of a dialog using either approach can appear almost exactly the same, there are distinct differences in the two approaches. These approaches are discussed in this chapter.

Property Sheet:

A Windows *property sheet* is a main window containing one or more overlapping windows. Each of the overlapping windows is called a *page*. The main window is called a property sheet, signifying a property sheet is used to set *properties*. In practice programmers use property sheets for a number of different reasons other than setting properties.

ooDialog implements property sheets through the *PropertySheetDialog* and the *PropertySheetPage* classes.

Control Dialogs:

Windows also has a style for dialogs that allows the dialog to work well as a dialog within a top-level dialog. The ooDialog framework has support for this type of dialog and calls them *control* dialogs because they function like a dialog control within the top-level dialog. These dialogs work well as the content for tab control pages. The implementation for control dialogs is primarily through the *ControlDialog* mixin class and the subclasses that inherit the **`ControlDialog`** class.

5.1. Control Dialogs

Windows supports a style for dialogs that allow the dialog to work well as a dialog within a top-level dialog. In essence a dialog with this style functions as a dialog control within the top-level dialog. Much of the behaviour and appearance and of these dialogs is controlled automatically by the operating system. The Windows programmer does not need to do much other than assign the proper style to the dialog. The ooDialog framework assigns the correct style to the control dialog and does not allow the Rexx programmer to change that.

The **`ControlDialog`** class is a mixin class that signals to the ooDialog framework that the *underlying* Windows dialog is to have the control dialog style. To use a control dialog in an ooDialog program, the programmer simply subclasses one of the 3 provided dialog classes that inherit the **`ControlDialog`** mixin class. These are the *RcControlDialog*, the *ResControlDialog* and the *UserControlDialog*. The 3 classes allow the programmer to base the dialog subclass on any one of the 3 basic ooDialog dialogs, a dialog based on a resource *script* file, dialog based on a binary compiled resource file, or a dialog where the programmer manually defines the dialog template in the program code. Because of this a control dialog is programmed in the same basic way as any other ooDialog dialog.

5.1.1. ControlDialog Class

The **`ControlDialog`** class is the mixin class that allows dialogs in ooDialog to be used as a dialog control. It is inherited by the **`RcControlDialog`**, **`ResControlDialog`**, and the **`UserControlDialog`** classes. To use a dialog as a control in a Rexx program, the programmer should pick one of the three basic types of control dialogs and write a subclass of that dialog:

- Subclass the *RcControlDialog* when the dialog template will be defined in a resource *script* file.
- Subclass the *ResControlDialog* when the dialog *template* will be loaded from a compiled binary resource file.

- Subclass the *UserControlDialog* when the dialog template will be defined using the *create...* methods.

5.1.1.1. endExecution

```
>>--endExecution(--+-------------+--)------------><
                   +--endLikeOk--+
```

The *endExecution* method must be used to finish the execution of all **`ControlDialog`** objects. Do not use any other mechanism to end the execution of any **`ControlDialog`** objects.

Arguments:

The single argument is:

endLikeOk [optional]

This argument specifies if the dialog is ending as though the user pressed the *Ok* button or the *Cancel* button. By default *endLikeOk* is false and the dialog is ended as though the user canceled the dialog. If *endLikeOk* is true, the dialog is ended as though the user pressed the *Ok* button.

Return value:

Returns the *IDOK* constant if the dialog was *wasActivated* and is closed as though the user pressed the *Ok* button. Otherwise, the IDCANCEL constant is returned.

Remarks:

When a dialog is used as a control in another dialog there are some operating system differences in how the *underlying* dialog behaves. The ooDialog framework ensures that the Rexx dialog object used as a dialog control behaves consistently with other dialogs in ooDialog. Closing a control dialog through some other mechanism than the *endExecution* method will bypass the internal processing used by the framework. This makes it likely that at some point the control dialog will not behave consistently with other dialogs.

Control dialogs are most useful as the pages in a *Tab* control. In Windows applications it is common practice to not create the underlying dialog for a tab page unless the user actually visits that page. If the Rexx control dialog, never has its underlying dialog created, then *endExecution* always returns the IDCANCEL constant and the Rexx dialog object is always ended as though the user had canceled the dialog.

Example:

This example shows the *ok* method of a main dialog that uses control dialogs as the pages of a tab control. If the user closes the main dialog with *ok* then all the control dialogs used as pages are ended using *endExecution* with an argument of true:

```
::method ok
    expose pages

    do page over pages
      if page \== .nil then page~endExecution(.true)
    end
    return self~ok:super
```

5.1.1.2. execute

```
>>--execute--------------------------------------><
```

All **`ControlDialog`** objects must be started through the *execute* method. This method causes the *underlying* dialog to be created. Do not use any other method to start control dialogs.

Arguments:

There are no argument to this method.

Return value:

Execute does not return any value.

Remarks:

In order to ensure that dialogs used as controls in main dialogs behave consistently, the programmer is required to start execution of all controld dialogs using the *execute* method and to end the dialog using the *endExecution* method. Failure to do so is not guaranteed to work.

Example:

This example has a control dialog of the **`.HistoryDlg`** subclass used as a page in a tab control. The dialog object is not instantiated until the user actually visits the page in the tab control. At that time, the dialog object is instantiated and then started executing throught the *execute* method:

```
::method activateHistory private unguarded
    expose historyDlg dlgRect pages

    reply

    historyDlg = .HistoryDlg~new(dlgRect, self)
    historyDlg~execute

    pages[4] = historyDlg

    self~postOnSelChange(10)
```

5.1.1.3. initUpdateListView

```
>>--initUpdateListView(--listViewID--+-----------+--)-----------><
                                     +--seconds--+
```

Causes the ooDialog framework to use an internal work around for a known Windows problem when a *ListView* control is used in the page of a *Tab* control.

Arguments:

The arguments are:

listViewID [required]

The resource ID of the list-view control in a page of a tab control, may be numeric or *symbolic*.

seconds [optional]

The positive number of seconds to pause before the list-view is redrawn. This should be a very small period expressed in decimal notation. Whole numbers are acceptable, but the pause will be too long to be of good use. The default value if this argument is omitted is .005.

Return value:

This method does not return a value.

Remarks:

There is a known problem in Windows when a list-view in used in the page of a tab control. If an item in the list-view is selected and the user selects another tab, then goes back to the first tab, the list-view sometimes does not redraw properly.

There is a work around for this problem and the ooDialog framework implements the work around internally so the Rexx programmer does not need to remember, or know, the details of the work around. If an application exhibits this problem, then use this method to fix it. The *initDialog* is the best place to invoke the method.

The solution to the problem is to have the list-view control redraw itself after the tab has been selected. The *seconds* argument can be used by the programmer to control the pause between the tab being selected and the list-view being redrawn. If the pause is too short, the list-view still may not redraw correctly. If the pause is too long, the user will notice the list-view being drawn twice. The *seconds* argument is only needed if the programmer is not satisfied with the appearance in a specific application.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example activates the internal work around for a list-view with the symbolic ID of IDC_LV_MAIN:

```
::method initDialog

  -- Initialize the internal fix for the list-view redrawing problem when a
  -- list-view is used in a tab control.
  self~initUpdateListView(IDC_LV_MAIN)
```

5.1.1.4. wasActivated (Attribute)

```
>>--wasActivated---------------------------------><

>>--wasActivated-=-trueFalse---------------------><
```

Reflects whether the *underlying* dialog for this control dialog was ever created.

wasActivated get:

The value of this attribute is true if the underlying dialog was created, otherwise it is false.

wasActivated set:

The programmer can not set the value of this attribute, it is set internally by the ooDialog framework.

Remarks:

Dialogs used as controls in other dialogs are most commonly used as a page in a tab control. In Windows it is a common practice to defer the creation of the dialog for the page until the tab page is actually visited by the user. This attribute allows the Rexx programmer to determine whether or not the underlying dialog has been created yet or not.

5.1.2. RcControlDialog Class

The **RcControlDialog** should be subclassed when the programmer wants to use a resource *script* to create the dialog *template* for his control dialog.

5.1.2.1. Method Table

The following table provides links to the documentation for the few methods and attributes used in working with **RcControlDialog** objects that differ from working with regular dialogs:

Table 5.1. Important RcControlDialog Methods

Method	Description
Class Methods	
new	Instantiates a new user control dialog.
Attributes	
ownerDialog	Reflects the dialog that owns this control dialog.
wasActivated	Reflects whether the *underlying* dialog for this control dialog was ever created.
Instance Methods	
endExecution	Must be used to finish the execution of all **RcControlDialog** objects.
execute	All **RcControlDialog** objects must be started through the *execute* method.
initUpdateListView	Initializes an internal work around for a problem with list-views that are used in a page of a tab control.

5.1.2.2. new (Class method)

```
>>--init(-scrpt-,-id-+---------+-+-----+-+--------+-+---------+-+---------+-)--><
                     +-,-data.-+ +-,-h-+ +-,-opts-+ +-,-xpect-+ +-,-oData-+
```

Instantiates a new **RcControlDialog** object.

Arguments:

The arguments when creating a new dialog instance of this class are:

scrpt [required]

The file name of the resource script containing the dialog template.

id [required]

The resource ID of the dialog template. This may be numeric or *symbolic*.

data. [optional]

A *dialog data stem* variable (don't forget the trailing period) that contains data used to initialize the underlying dialog's controls.

h [optional]

The name of a *file*) containing symbolic ID defines for resource IDs.

opts [optional]

Zero or more of the following keywords, separated by blanks:

CENTER

The dialog is to be positioned in the center of the screen.

CONNECTBUTTONS

Each push *Button* in the underlying dialog has the CLICKED *event* notification connected automatically to a method in the Rexx dialog object. This is the same as using the *connectButtonEvent*() method for the CLICKED notification. The name for the method is generated automatically by the ooDialog framework. The method name is the button label with all spaces, ampersands, colons, and trailing ... characters removed.

CONNECTRADIOS

Similar to CONNECTBUTTONS, this option connects the CLICKED event notification from each *RadioButton* button to a method in the Rexx dialog object. Again, this is the same as using the *connectButtonEvent* method. For radio buttons, the generated

CONNECTCHECKS

Exactly the same as CONNECTRADIOS, for check *CheckBox* controls. The object method name is generated in the same way as it is for radio buttons. That is, the method name is the button label, with all spaces, ampersands, colons, and trailing ... characters removed. Which is then **prepended** with the text **`ID`**.

xpect [optional]

This is the maximum number of dialog controls expected in the dialog template. It serves the same purpose as the *expected* argument in the *create*() method of the **`UserDialog`**. The default value for this argument is 200.

oData [optional]

Used to specifiy the *owner* dialog of this *ControlDialog*. All control dialogs *must* have the *ownerDialog* attribute set to the Rexx dialog that owns the control dialog, before the *underlying* control dialog is created. In addition, the underlying owner dialog must be created before the underlying control dialog can be created.

The *oData* argument can be used to set the *ownerDialog* attribute when this control dialog object is instantiated. The Rexx owner dialog's underlying dialog does not have to be created when the attribute is set. The programmer just needs to ensure that the underlying dialog *has* been created when the *execute* method is invoked.

Remarks:

Normally a programmer does not instantiate a **`RcControlDialog`** directly, but rather creates a subclass of a **`RcControlDialog`** and instantiates the subclass. Recall that in ooRexx, when a new object is instantiated, the *new* method invokes the *init* method of the object using the arguments passed to the *new* method. So, the arguments of the *new* method are also the arguments of the *init* method.

If the programmer over-rides the *init* method in the subclass of the **`RcControlDialog`**, care must be taken to invoke the superclass *init* method with the correct arguments.

Example:

This example shows the instantiation of a **`RcControlDialog`** subclass, (which is really no different than instantiating a *RcDialog* subclass.) Note that the *ownerDialog* attribute is set after the invocation of the *new* method. This is sometimes easier than using a lot of omitted arguments for a **`RcControlDialog`**.

```
...

dlg3 = .ProgressBarDlg~new("rc\PropertySheetDemo.rc", IDD_PROGRESSBAR_DLG)

-- Create the main dialog.
dlg = .NewControlsDialog~new('rc\PropertySheetDemo.dll', IDD_NEWCONTROLS_DLG)
```

```
  dlg3~ownerDialog = dlg

::requires "ooDialog.cls"

...
```

5.1.3. ResControlDialog Class

Subclass the **ResControlDialog** to use a control dialog in an application where the dialog *template* comes from a compiled binary resource file.

5.1.3.1. Method Table

The following table provides links to the documentation for the few methods and attributes used in working with **ResControlDialog** objects that differ from working with regular dialogs:

Table 5.2. Important ResControlDialog Methods

Method	Description
Class Methods	
new	Instantiates a new user control dialog.
Attributes	
ownerDialog	Reflects the dialog that owns this control dialog.
wasActivated	Reflects whether the *underlying* dialog for this control dialog was ever created..
Instance Methods	
endExecution	Must be used to finish the execution of all **ResControlDialog** objects.
execute	All **ResControlDialog** objects must be started through the *execute* method.
initUpdateListView	Initializes an internal work around for a problem with list-views that are used in a page of a tab control.

5.1.3.2. new (Class method)

```
>>--new(--module--,--id--+-------------+--+----------+--+-------------+--)-----><
                         +-,--dlgData.-+  +-,--hFile-+  +-,-ownerData-+
```

Instantiates a new **ResControlDialog** object. The dialog template for the object is taken from the specified module, which is usually a resource only DLL.

Arguments:

The arguments when creating a new dialog instance of this class are:

module [required]

The file name of the executable module (a DLL or EXE) in which the resource (the compiled dialog template) is located.

id [required]

The resource ID of the dialog template. This may be numeric or *symbolic*. The resource ID is assigned to the dialog template when it was compiled.

dlgData. [optional]

A *dialog data stem* variable (don't forget the trailing period) that contains data used to initialize the underlying dialog's controls.

hFile [optional]

The name of a *file*) containing symbolic ID defines for resource IDs.

ownerData [optional]

Used to specifiy the *owner* dialog of this *ControlDialog*. All control dialogs *must* have the *ownerDialog* attribute set to the Rexx dialog that owns the control dialog, before the *underlying* control dialog is created. In addition, the underlying owner dialog must be created before the underlying control dialog can be created.

The *ownerData* argument can be used to set the *ownerDialog* attribute when this control dialog object is instantiated. The Rexx owner dialog's underlying dialog does not have to be created when the attribute is set. The programmer just needs to ensure that the underlying dialog *has* been created when the *execute* method is invoked.

Remarks:

Normally a programmer does not instantiate a **`ResControlDialog`** directly, but rather creates a subclass of a **`ResControlDialog`** and instantiates the subclass. Recall that in ooRexx, when a new object is instantiated, the *new* method invokes the *init*() method of the object, using the method arguments of the *new*() method. So, the arguments of the *new* method are also the arguments of the *init* method.

Therefore, if the programmer over-rides the *init* method in the subclass of the **`ResControlDialog`**, care must be taken to invoke the superclass *init* method with the correct arguments.

When a **`ResControlDialog`**, or subclass, is instantiated, and the file specified in the *module* argument can not be found, or can not be loaded as an executable module, a message box will pop up informing the user of the problem.

Example:

This example shows the instantiation of a **`ResControlDialog`** subclass, (which is really no different than instantiating a *ResDialog* subclass.) Note that the *ownerDialog* attribute is set after the invocation of the *new* method. This is sometimes easier than using a lot of omitted arguments for a **`ResControlDialog`**.

```
  ...

  dlg4 = .TrackBarDlg~new("rc\PropertySheetDemo.dll", IDD_TRACKBAR_DLG)

  -- Create the main dialog.
  dlg = .NewControlsDialog~new('rc\PropertySheetDemo.dll', IDD_NEWCONTROLS_DLG)

  dlg4~ownerDialog = dlg

::requires "ooDialog.cls"

...
```

5.1.4. UserControlDialog Class

The **`UserControlDialog`** should be used when the programmer wants to define the dialog *template* for the control dialog by using the *create...* methods.

5.1.4.1. Method Table

The following table provides links to the documentation for the few methods and attributes used in working with **UserControlDialog** objects that differ from working with regular dialogs:

Table 5.3. Important UserControlDialog Methods

Method	Description
Class Methods	
new	Instantiates a new user control dialog.
Attributes	
ownerDialog	Reflects the dialog that owns this control dialog.
wasActivated	Reflects whether the *underlying* dialog for this control dialog was ever created..
Instance Methods	
endExecution	
execute	
initUpdateListView	Initializes an internal work around for a problem with list-views that are used in a page of a tab control.

5.1.4.2. new (Class method)

```
>>--new(--+----------+--+--------------+--+-------------+--)---><
          +-dlgData.-+  +-,--headerFile-+  +-,-ownerData-+
```

Instantiates a new **UserControlDialog** object.

Arguments:

The arguments are:

dlgData. [optional]

A *dialog data stem* variable (don't forget the trailing period) that contains data used to initialize the underlying dialog's controls.

hFile [optional]

The name of a *file*) containing symbolic ID defines for resource IDs.

ownerData [optional]

Used to specifiy the *owner* dialog of this *ControlDialog*. All control dialogs *must* have the *ownerDialog* attribute set to the Rexx dialog that owns the control dialog, before the *underlying* control dialog is created. In addition, the underlying owner dialog must be created before the underlying control dialog can be created.

The *ownerData* argument can be used to set the *ownerDialog* attribute when this control dialog object is instantiated. The Rexx owner dialog's underlying dialog does not have to be created when the attribute is set. The programmer just needs to ensure that the underlying dialog *has* been created when the *execute* method is invoked.

Remarks:

Normally a programmer does not instantiate a **UserDialog** directly, but rather creates a subclass of a **UserDialog** and instantiates the subclass. Recall that in ooRexx, when a new object is instantiated, the *new* method invokes the *init*() method of the object, using the arguments of the *new*() method. Therefore, the arguments of the *new* method are also the arguments of the *init* method.

If the programmer over-rides the *init* method in the subclass of the **`UserDialog`**, care must be taken to invoke the superclass *init* method with the correct arguments.

Example:

This sample shows a method in the main dialog that instantiates a subclass of a **`UserControlDialog`** and shows the init() method of that subclass. Note the proper way that the **`PatientDlg`** object initializes its superclass.

```
::method activatePatient private unguarded
    expose patientDlg dlgRect pages

    patientDlg = .PatientDlg~new(dlgRect, self)
    patientDlg~execute

    pages[1] = patientdlg

    ...


::class 'PatientDlg' subclass UserControlDialog

::method init
    use arg r, parent

    self~init:super( , , parent)
    self~create(r~left, r~top, r~right, r~bottom)
    ...
```

5.2. PropertySheetDialog Class

The Windows *property sheet* is a dialog that contains one or more overlapping child windows called *pages*. Each page normally contains a group of related dialog controls. Each page has a tab that the user can select to bring the page to the foreground of the property sheet. The main dialog itself is the property sheet. The name *property sheet* indicates that Microsoft's original purpose for the property sheet was to set properties. However, property sheet dialogs are useful in any situation where the programmer needs to place a number of dialog controls without using a huge dialog. In practice, programmers have not restricted the use of property sheet dialogs to strictly setting properties.

There is a special type of property sheet called a *wizard*. Wizards are designed to show the pages one at a time in a sequence that is controlled by the application. Instead of selecting from a group of pages by clicking a tab, users navigate forward and backward through the sequence, one page at a time, by clicking Next or Back buttons located at the bottom of the wizard.

The ooDialog framework provides the **`PropertySheetDialog`** and *PropertySheetPage* classes that allow the programmer to create property sheet and wizard dialogs. Unlike most other ooDialog dialogs, the operating system manages much of the property sheet dialog itself. For instance, the operating system executes and shows the page dialogs itself. The ooDialog programmer does not invoke the *execute* method for the page dialogs.

5.2.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with **`PropertySheetDialog`** objects.

Table 5.4. PropertySheetDialog Methods

Method	Description
Constant Methods	
ID_PSREBOOTSYSTEM	Possible return value from the *execute* or *getResult* methods. Indicates a page sent the reboot system message to the property sheet. The computer must be restarted for the user's changes to take effect. The application is responsible for rebooting the computer.
ID_PSRESTARTWINDOWS	Possible return value from the *execute* or *getResult* methods. Indicates a page sent the restart Window message to the property sheet. Microsoft Windows must be restarted for the user's changes to take effect. The application is responsible for restarting Windows.
MAXPROPPAGES	The maximum number of pages a property sheet dialog can have. This is an operating system restriction.
Class Methods	
new	Instantiates a new **`PropertySheetDialog`** object.
Attributes	
appIcon	Reflects the value of the *dialog icon* for the property sheet dialog.
caption	Reflects the caption, or title, of the property sheet dialog.
header	Specifies the bitmap that will be placed at the right side of an interior page's header area in a Wizard dialog. The attribute is only used for the Wizard style property sheet dialogs.
imageList	Used to assign icons for the tabs of the tab control in a property sheet dialog.
pages	Reflects the dialog objects used as the pages of this property sheet dialog.
resources	Can be set to a *ResourceImage* object to supply some or all of the resources used by the property sheet dialog.
startPage	Reflects the index of the page that will be selected and visible when the property sheet dialog is first shown on the screen.
watermark	Reflects the watermark bitmap image that will be used with a Wizard97 property sheet dialog.
Instance Methods	
addPage	Adds a page to the property sheet at the end of the current pages.
apply	Simulates the selection of the Apply button, indicating that one or more pages have changed and the changes need to be validated and recorded.
cancelToClose	Used when changes made since the most recent APPLY notification cannot be canceled.
changed	Informs the underlying property sheet dialog that information in a page has changed.
enableWizButtons	Enables or disables buttons in an Aero wizard.
execute	Starts the execution of the underlying, *modal*, property sheet dialog.
getCurrentPageHwnd	Retrieves the window *handle* of the *underlying* dialog of the current page in this property sheet dialog.
getPage	Gets the Rexx dialog object for the page specified by *index*.

Method	Description
getResult	Returns a keyword indicating the result of executing the **`PropertySheetDialog`**.
getTabControl	Retrieves the Rexx tab control object for the tab control of the property sheet dialog.
hwndToIndex	Returns the one-based index of the property sheet page specified by its window handle.
idToIndex	Takes a property sheet page ID and returns its one-based page index.
indexToHwnd	Given the one-based index to a page, returns the window handle of the underlying page dialog.
indexToID	Given the index of a page, returns its ID.
indexToPage	Returns the PROPSHEETPAGE *handle* for the specified page index.
initDialog	Invoked by the ooDialog framework when the underlying property sheet dialog has first been created.
insertPage	Inserts a new page into the property sheet dialog at the specified index.
pageToIndex	Takes the handle of a *indexToPage* and returns the one-based index of the page.
popup	Begins the execution of a *modeless* property sheet dialog and returns immediately.
popupAsChild	Starts a *modeless* property sheet dialog executing and returns immediately. The property sheet dialog is closed automatically when its parent is closed
pressButton	Pushes the specified property sheet button programmatically.
querySiblings	Causes the underlying property sheet dialog to send the *queryFromSibling* notification, with the specified arguments to each page of the of the property sheet dialog.
rebootSystem	Invoked to notify the property sheet dialog that the system needs to be rebooted for changes to take effect.
recalcPageSizes	Recalculates the page size of a standard or wizard property sheet after pages have been added or removed.
restartWindows	Invoked to notify the property sheet dialog that Microsoft Windows needs to be restarted for changes to take effect.
removePage	Removes a page from this property sheet.
setButtonText	Sets the text of the specified button in an Aero wizard.
setCurSel	Activates the specified page in a property sheet.
setCurSelByID	Activates the page specified by *id* in this property sheet.
setFinishText	Sets the text of the Finish button in a wizard, shows and enables the button, and hides the Next and Back buttons.
setHeaderSubTitle	Sets or resets the text for the header subtitle in a wizard property sheet dialog.
setHeaderTitle	Sets or resets the text for the header in a wizard property sheet dialog.
setNextText	Sets the text of the Next button in an Aero wizard.
setTitle	Sets the title for a property sheet dialog.

Method	Description
setWizButtons	Enables or disables the Back, Next, and Finish buttons in a wizard.
showWizButtons	Shows or hides buttons in an Aero wizard.
unchanged	Informs this property sheet that information in a page has reverted to the previously saved state.

5.2.2. new (Class method)

```
>>--new(--pages--+--------+--+---------+--+---------+--)-----------------------><
                 +-,-opts-+  +-,-title-+  +-,-hFile-+
```

Instantiates a new **`PropertySheetDialog`** object.

Arguments:

The arguments when creating a new dialog instance of this class are:

pages [required]

An array of *PropertySheetPage* dialogs to be used as the pages of the property sheet. The number of the property sheet page dialogs and the order of the dialogs determines the number and order of the pages in the property sheet.

opts [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant:

AEROWIZARD	PROPTITLE	WIZARDCONTEXTHELP
HASHELP	RESIZABLE	WIZARDHASFINISH
NOAPPLYNOW	RTLREADING	WIZARDLITE
NOCONTEXTHELP	USEPAGELANG	
NOMARGIN	WIZARD97	

AEROWIZARD

Creates a wizard property sheet that uses the newer Aero style. This keyword is not valid if the operating system is not Vista or later. A syntax condition is raised if this keyword is used on Windows XP or earlier.

HASHELP

Permits property sheet pages, subclasses of the *PropertySheetPage* class, to display a *Help* button. The HASHELP keyword must also be used in the page's *opts* argument when the page is instantiated. If any of the initial property sheet pages enable a Help button, the operating system will automatically enables this feature in the property sheet dialog. If none of the initial pages enable a Help button, the HASHELP keyword must be explicitly set in order to have Help buttons on any pages that might be added later. This keyword is not supported if the AEROWIZARD style is used.

NOAPPLYNOW

Removes the Apply button. This keyword is not supported with the AEROWIZARD style.

NOCONTEXTHELP

Removes the context-sensitive Help button ("?"), which is usually present on the caption bar of property sheets. This flag is not valid for wizards.

NOMARGIN
: Specifies that no margin is inserted between the page and the frame. Must be used in combination with AEROWIZARD.

PROPTITLE
: Adjusts the title in the title bar of the property sheet. The title takes the appropriate form for the Microsoft Windows version. In more recent versions of Windows, the title is the string specified by the *caption* attribute followed by the string *Properties*. In older versions of Windows, the title is the string *Properties for*, followed by the string specified by the *caption* attribute This keyword is not supported with the AEROWIZARD style.

RESIZABLE
: Allows the wizard to be resized by the user. Maximize and minimize buttons appear in the wizard's frame and the frame is sizable. To use this keyword, the AEROWIZARD keyword must also be set.

RTLREADING
: Reverses the direction in which the tile of the property dialog is displayed. Normal windows display all text, including the title, left-to-right (LTR). For languages such as Hebrew or Arabic that read right-to-left (RTL), a window can be mirrored and all text will be displayed RTL. If RTLREADING is set, the title will instead read RTL in a normal parent window and LTR in a mirrored parent window.

USEPAGELANG
: Specifies that the language for the property sheet dialog will be taken from the first page's resource. That page must be specified by resource identifier. This keyword is included for completeness, it will have no effect in the current ooDialog implementation. It is possible that a future enhancement to ooDialog will make this keyword relevant

WIZARD97
: Creates a Wizard97-style property sheet that allows a header and/or watermark bitmap to be displayed in the background. This keyword is not supported with the AEROWIZARD style.

WIZARDCONTEXTHELP
: Adds a context-sensitive Help button ("?"), which is usually absent from the caption bar of a wizard. This keyword is not supported with the AEROWIZARD style.

WIZARDHASFINISH
: Always displays the Finish button on the wizard. One of the other WIZARDxx keywords must also be used.

WIZARDLITE
: Uses the Wizard-lite style. This style is similar in appearance to WIZARD97. There are few restrictions on how the pages are formatted. For instance, there are no enforced borders, and the WIZARDLITE style does not paint the watermark and header bitmaps the way WIZARD97 does. This keyword is not supported with the AEROWIZARD style.

title [optional]
: Sets the *caption* attribute to the text specified.

hFile [optional]
: A file, (often called a header *file*,) defining *symbolic* IDs for resources. The symbolic IDs defined within the file will be added to the *constDir* directory.

Remarks:

`PropertySheetDialog` dialogs are a little different than most ooDialog dialogs in that their life-cycle is controlled by the operating system rather than the ooDialog framework. In general the programmer customizes these dialogs by using the proper keywords in the *opts* argument and by setting the attributes of the dialog object, then starting the dialog through *execute* or *popup*. Most of the programming work for a property sheet dialog is in the dialogs that are used as the pages for the property sheet.

Example:

This example sets up the 5 dialogs to be used as the pages of the property sheet dialog and then instantiates that dialog:

```
  -- Create the 5 dialog pages.
  p1 = .ListViewDlg~new("rc\PropertySheetDemo.rc", IDD_LISTVIEW_DLG)
  p2 = .TreeViewDlg~new("rc\PropertySheetDemo.rc", IDD_TREEVIEW_DLG)
  p3 = .ProgressBarDlg~new("rc\PropertySheetDemo.rc", IDD_PROGRESSBAR_DLG)
  p4 = .TrackBarDlg~new("rc\PropertySheetDemo.rc", IDD_TRACKBAR_DLG)
  p5 = .TabDlg~new("rc\PropertySheetDemo.rc", IDD_TAB_DLG)

  -- Create the PropertySheetDialog using an array of the 5 dialog pages.  The
  -- order of the pages in the array will be the order of the pages in the tab
  -- control of the property sheet.
  pages = .array~of(p1, p2, p3, p4, p5)
  propDlg = .PropertySheetDemoDlg~new(pages, "NOAPPLYNOW", "ooRexx Property Sheet with
 Controls")

  -- Do any customization of the property sheet dialog by setting the values of
  -- the appropriate attributes.

...
```

5.2.3. appIcon (Attribute)

```
>>--appIcon--------------------------------------><

>>--appIcon=icon---------------------------------><
```

Reflects the value of the *dialog icon* for the property sheet dialog.

appIcon get:

The value of the *appIcon* attribute will be the **`.nil`** object if the programmer has not specified a dialog icon. Otherwise the value is that specified.

appIcon set:

The *appIcon* can be set either using an *Image* object, or a resource ID, numeric or *symbolic*.

Remarks:

When the *appIcon* is specified as an **`Image`** object, then the image must actually be an icon. It can not be a bitmap, or cursor, etc..

When the *appIcon* attribute is specified as a resource ID, then the image must be a resource in the resource *ResourceImage* specified by the *resources* attribute. When the property sheet dialog is shown, if an icon can not be loaded from the resource image, the property sheet dialog will not have a dialog icon.

Details

Raises syntax errors when incorrect usage in setting the attribute is detected. If the *appIcon* attribute is set to a numeric value, the ooDialog framework can not detect whether the number is correct or not. If the number, (or if the number a symbolic ID resolves to,) is not a resource ID of a resource in the *resources* resource image, then the property sheet dialog will not have a dialog icon when it is shown.

Example:

This example sets the dialog icon for a property sheet dialog:

```
-- For the application icon of the dialog we will use the ooRexx icon which is
-- available for general use from oodialog.dll.
propDlg~resources = .ResourceImage~new("oodialog.dll", propDlg)
propDlg~appIcon = "IDI_DLG_OOREXX"
```

5.2.4. caption (Attribute)

```
>>--caption--------------------------------------><

>>--caption=varName------------------------------><
```

Reflects the caption, or title, of the property sheet dialog.

caption get:

Returns the string to be used for the property sheet dialog, or the **`.nil`** object if no caption has been set.

caption set:

Sets the value the property sheet dialog will use for its title bar when the *underlying* dialog is created.

Remarks:

This is not a *dynamic* attribute. I.e., if the underlying property sheet has its title bar caption changed, say through the *setTitle* method, the value of the *caption* attribute will not change. Likewise, if the programmer were to change the value of this attribute after the underlying dialog was created, it would not change the title bar of the dialog. At the time the underlying dialog is created, this attribute is used to set the dialog's title bar. Other than that, the attribute does nothing.

Details

Raises syntax errors when incorrect arguments are detected.

5.2.5. header (Attribute)

```
>>--header---------------------------------------><

>>--header=varName-------------------------------><
```

The *header* attribute specifies the bitmap that will be placed at the right side of an interior page's header area in a Wizard dialog. The *header* attribute is only used for the Wizard style property sheet dialogs.

header get:

The value of the *header* attribute is the **.nil** object until it has been set by the programmer. Unless the dialog is a Wizard property sheet dialog, it will always be the **.nil** object.

header set:

The *header* attribute should be set either to an *Image* object of the bitmap to display in the header area of a Wizard dialog, or to the resource ID of the bitmap to be used. Do not set the *header* attribute if the dialog is not a Wizard.

Remarks:

The bitmap for the header area can either be a stand alone bitmap loaded as an **Image** object, or it can be a compiled binary image located in the resource section of an executable file. If the bitmap is within the resource section of an executable module, then the *resources* attribute of this dialog object is set to the executable and the *header* attribute is set to the resource ID of the bitmap.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example shows the header attribute of a Wizard dialog being set to an **Image** object containing the desired bitmap to be used in the header area of the Wizard:

```
-- Create the property sheet using the dialog pages and set its attributes.
wizDlg = .TicketWizard97~new(pages, "Wizard97", "Let's Go To The Movies")

wizDlg~header = .Image~getImage("rc\ticketWizardTheater.bmp")
wizDlg~watermark = .Image~getImage("rc\ticketWizardRexxLA.bmp")
```

5.2.6. imageList (Attribute)

```
>>--imageList------------------------------------><

>>--imageList=varName----------------------------><
```

The *imagelist* attribute is used to assign icons for the tabs of the tab control in a property sheet dialog.

imageList get:

The value of the *imageList* attribute is an *ImageList* object if one has been assigned, otherwise it is the **.nil** object.

imageList set:

When an *ImageList* object is assigned to the *imageList* attribute, the icons in the imagelist are used as the icons in the tabs of the pages in the property sheet. The icons are assigned in order to the pages, the first icon in the imagelist is assigned to the first page, the second icon to the second page, etc..

Remarks:

An icon for the tab of a page can also be assigned through the *tabIcon* attribute of the *PropertySheetPage* dialog object for each page in a property sheet dialog. However, assigning an image list to the *imageList* attribute takes precedence over the *tabIcon* attribute of a property sheet page dialog. If an image list is assigned to the *imageList* attribute of the property sheet dialog, the *tabIcon* attribute of the property sheet page dialog is not checked.

Details

Raises syntax errors when incorrect arguments are detected.

5.2.7. pages (Attribute)

```
>>--pages----------------------------------------><

>>--pages=varName--------------------------------><
```

Reflects the dialog objects used as the pages of this property sheet dialog.

pages get:

Returns an array of the dialog objects used for the pages of the property sheet dialog. The dialog object at index 1 is the dialog for the first page in the property sheet. The dialog object at index 2 is the dialog for the second page, etc..

pages set:

The programmer can not set this attribute, it is set by the ooDialog framework.

Remarks:

The programmer can not manipulate the pages in the property sheet dialog by manipulating the array that is the value of this attrbiute. I.e., if the programmer removes an index of the array, it does not remove that page in the property sheet, if the programmer adds an index to the array, it does not add a page to the property sheet dialog. Page manipulation must be done through the instance methods of the property sheet dialog, for instance, the *insertPage* or *removePage* methods.

5.2.8. resources (Attribute)

```
>>--resources------------------------------------><

>>--resources-=-image----------------------------><
```

The *resources* attribute can be set to a *ResourceImage* object to supply some or all of the resources used by the **`PropertySheetDialog`**.

resources get:

Returns the **`ResourceImage`** object if this attribute has been set by the programmer or the **`.nil`** object if the attrbute has not been set.

resources set:

This attribute can only be set to a *ResourceImage* object.

Remarks:

Many of the resources used by a property sheet dialog can be located within the resource section of an executable module. (Recall that in Windows both an executable file and a DLL are executable modules.) Resources such as the *appIcon*, *header*, *imageList*, etc.. can all be located in the **`ResourceImage`** that is the value of the *resources* attribute.

These resources are then all specified by their resource ID in the resource image. Since the *resources* attribute can only have one value, this implies that all resources specified by resource ID must reside within the same resource image.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example shows the assignment of a **`ResourceImage`** object to the *resources* attribute and specifying the resource ID to use for the appIcon of the property sheet dialog:

```
-- For the application icon of the dialog we will use the ooRexx icon which is
-- available for general use from oodialog.dll.

propDlg~resources = .ResourceImage~new("oodialog.dll", propDlg)
propDlg~appIcon = IDI_DLG_OOREXX
```

5.2.9. startPage (Attribute)

```
>>--startPage------------------------------------><

>>--startPage=varName----------------------------><
```

Reflects the index of the page that will be selected and visible when the property sheet dialog is first shown on the screen.

startPage get:

Returns the one-based index of the page that will be visible when the property sheet dialog is started.

startPage set:

By default the *startPage* attribute is set to 1 so that the first page in the property sheet is shown first. The first page to be shown can be changed by setting this attribute to the index of some other page.

Remarks:

The *startPage* attribute must be set within the range of 1 to *MAXPROPPAGES* or a syntax condition will be raised. If *startPage* is set to an index larger than the number of pages, then the first page will be shown.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example changes the first page to be shown from the default to page 3:

```
propDlg~startPage = 3
```

5.2.10. watermark (Attribute)

```
>>--watermark------------------------------------><

>>--watermark=varName----------------------------><
```

Reflects the watermark bitmap image that will be used with a Wizard97 property sheet dialog.

watermark get:

Returns the bitmap image, or resource ID of the bitmap image, that will be used for the watermark. The value of this attribute is the **`.ni`** object if the attrbute has not been set or the property sheet dialog is not a Wizard97 property sheet dialog.

watermark set:

The *watermark* attribute should be set either to an *Image* object of the bitmap to display as the watermark of a Wizard dialog, or to the resource ID of the bitmap to be used. Do not set the *watermark* attribute if the dialog is not a Wizard97 property sheet dialog.

Remarks:

The bitmap for the watermark can either be a stand alone bitmap loaded as an **`Image`** object, or it can be a compiled binary image located in the resource section of an executable file. If the bitmap is within the resource section of an executable module, then the *resources* attribute of this dialog object is set to the executable and the *watermark* attribute is set to the resource ID of the bitmap.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example creates a Wizard97 property sheet dialog and sets the watermark bitmap to an **`Image`** object. The bitmap will be loaded from the **`ticketWizardRexxLA.bmp`** bitmap file:

```
wizDlg = .TicketWizard97~new(pages, "Wizard97", "Let's Go To The Movies")

wizDlg~header = .Image~getImage("rc\ticketWizardTheater.bmp")
wizDlg~watermark = .Image~getImage("rc\ticketWizardRexxLA.bmp")
```

5.2.11. addPage

```
>>--addPage(--pageDlg--+---------------+--)------><
                       +-,-isExterior--+
```

Adds a page to the property sheet at the end of the current pages.

Arguments:

The arguments are:

pageDlg [required]

The Rexx dialog object to use as the page being added.

isExterior [optional]

Specifies if the added page is an exterior page of a wizard property sheet dialog. If the property sheet dialog is not a wizard, this argument is ignored.

Return value:

Returns true on success, otherwise false.

Remarks:

Syntax conditions are raised if the *page* argument is not correct. The argument must be a *PropertySheetPage* dialog object.

This method must not be invoked from within the event handling methods, *apply*, *killActive*, *reset*, or *setActive* of the *PropertySheetPage* class. However, you can add or remove pages during the *wizBack* or *wizNext* event handling methods, provided the correct index to specify the new page is returned from those event handlers.

The dialog object must not have already been *used* as a property sheet page. In addition to adding or inserting pages in a property sheet, pages can be removed from a property sheet. When the page is removed, the operating system destroys the underlying Windows dialog. Although the Rexx dialog object is still active, that object can not be reinserted into a property sheet as a page.

It is possible that the user never *visited* a page before it was removed. In this case the Rexx dialog object could be used to add or insert a new page at some later point, because the underlying Windows dialog would never have been created. The *wasActivated* attribute can be used to test for this condition. However, it is probably simplier to always instantiate a new Rexx dialog object for each page added to a property sheet.

The Windows operating system restricts the maximum number of pages that a property sheet can hold. The MAXPROPPAGES constant of the *PropertySheetDialog* reflects this number. If the programmer tries to insert or add a page past this number, a condition is raised.

5.2.12. apply

```
>>--apply----------------------------------------><
```

Simulates the selection of the Apply button, indicating that one or more pages have changed and the changes need to be validated and recorded.

Arguments:
There are no arguments for this method.

Return value:
Returns 0, always.

5.2.13. cancelToClose

```
>>--cancelToClose--------------------------------><
```

Used when changes made since the most recent APPLY notification cannot be canceled.

Arguments:
There are no arguments for this method.

Return value:
Returns 0, always.

Remarks:
cancelToClose disables the *Cancel* button and changes the text of the OK button to *Close*.

5.2.14. changed

```
>>--changed(--pageDlg--)-------------------------><
```

Informs the underlying property sheet dialog that information in a page has changed.

Arguments:

The single argument is:

pageDlg

The Rexx dialog object that is the page that is changed.

Return value:

Returns 0, always.

Remarks:

The underlying property sheet enables the *Apply* button.

Details

Raises syntax errors when incorrect usage is detected.

5.2.15. enableWizButtons

```
>>--enableWizButtons(--+--------+--+---------------+--)--------><
                       +--opts--+  +-,-optsButtons--+
```

Enables or disables buttons in an Aero wizard. *Vista* or later only. This method does nothing if this property sheet is not an Aero wizard.

Arguments:

The arguments are:

opts [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. These specify which property sheet buttons are to be enabled. If a button value is included in both this argument and *optsButton* then it is enabled.

BACK	FINISH
CANCEL	NEXT

BACK

The Back button.

CANCEL

The Cancel button.

FINISH

The Finish button.

NEXT

The Next button.

opts [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. These specify which property sheet buttons are affected by this method invocation. If a button value appears in this parameter but not in *opts*, the button is disabled.

BACK	FINISH
CANCEL	NEXT

BACK

The Back button.

CANCEL
: The Cancel button.

FINISH
: The Finish button.

NEXT
: The Next button.

Return value:

Returns true if this is an Aero Wizard property sheet on Vista on later, otherwise false.

Remarks:

This method only works for Aero Wizards. This method requires Vista or later, a condition is raised if the OS is not Vista or later. This method has no effect if the property sheet is not an Aero wizard.

Details

Raises syntax errors when incorrect usage is detected.

5.2.16. execute

```
>>--execute(--+---------+--)---------------------><
              +--owner--+
```

Starts the execution of the underlying, *modal*, property sheet dialog. This method does not return until the user has closed the property sheet dialog.

Arguments:

The single argument is:

owner [optional]
: The Rexx dialog object whose underlying dialog is the owner dialog of this property sheet dialog. Typically modal dialogs have an owner dialog. If this argument is omitted, then no owner dialog is set.

Return value:

Returns a positive number if successful and -1 on error. It is possible that the return will be either one of the *constants* ID_PSRESTARTWINDOWS or ID_PSREBOOTSYSTEM.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

5.2.17. getCurrentPageHwnd

```
>>--getCurrentPageHwnd---------------------------><
```

Retrieves the window *handle* of the *underlying* dialog of the current page in this property sheet dialog.

Arguments:

There are no arguments to this method.

Return value:

Returns the current pages window handel.

5.2.18. getPage

```
>>--getPage(--index--)---------------------------><
```

Gets the Rexx dialog object for the page specified by *index*.

Arguments:

The single argument is:

index [required]

The one-based index of the page in the property sheet whose Rexx dialog object should be retrieved.

Return value:

The Rexx dialog object for the page specified

Details

Raises syntax errors when incorrect usage is detected.

5.2.19. getResult

```
>>--getResult------------------------------------><
```

Returns a keyword indicating the result of executing the **`PropertySheetDialog`**.

Arguments:

There are no arguments for this method.

Return value:

Returns one of the following keywords to indicate the result of executing the property sheet dialog:

NotFinished

The property sheet dialog has not been closed yet.

ExecutionErr

The property sheet dialog did not execute because of an error.

RebootSystem

A page sent the reboot system message to the property sheet. The computer must be restarted for the user's changes to take effect.

RestartWindows

A page sent restart Windows message to the property sheet. Microsoft Windows must be restarted for the user's changes to take effect.

ClosedCancel

The user canceled the dialog

ClosedOk

The user closed the dialog with ok.

Unknown
: This return should never happen. It would indicate the internal state of the ooDialog framework is inconsistent.

5.2.20. getTabControl

```
>>--getTabControl--------------------------------><
```

Retrieves the Rexx tab control object for the tab control of the property sheet dialog.

Arguments:

There are no argument for this method.

Return value:

Returns the Rexx *Tab* control for the underlying Windows tab control in the property sheet dialog

5.2.21. hwndToIndex

```
>>--hwndToIndex(--hwnd--)------------------------><
```

Returns the one-based index of the property sheet page specified by *hwnd*, its window handle.

Arguments:

The single argument is:

hwnd [required]
: The *hwndhandle* whose page index is needed.

Return value:

Returns the one-based index of the page on success, otherwise zero.

5.2.22. idToIndex

```
>>--idToIndex(--id--)----------------------------><
```

Takes a property sheet page ID and returns its one-based page index.

Arguments:

The single argument is:

ID [required]
: The property sheet page *indexToID*.

Return value:

The one-based index of the page on success, or 0 on error.

5.2.23. indexToHwnd

```
>>--indexToHwnd(--index--)-----------------------><
```

Given the one-based index to a page, returns the window handle of the underlying page dialog.

Arguments:

The single argument is:

index [required]

The one-based page index.

Return value:

The window handle for the page.

5.2.24. indexToID

```
>>--indexToID(--index--)------------------------><
```

Given the index of a page, returns its ID.

Arguments:

The single argument is :

index [required]

The one-based index of the page. The special values 0 and -1 are also acceptable.

Return value:

The proper page ID for the index.

Remarks:

Page IDs are different depending on the type of the dialog page, *UserPSPDialog*, *ResPSPDialog*, etc.. In order to get the correct ID, the Rexx programmer must use this method.

There are also two special case values for *index*. The page ID would most commonly be used in the *setActive*, *wizBack*, or *wizNext* event notification methods that signal a page is being changed. To accept the page change, the event handler returns 0, and to cancel the page change the event handler returns -1. So, 0 and -1 are acceptable here and return the proper value for those methods.

Details

Raises syntax errors when incorrect usage is detected.

5.2.25. indexToPage

```
>>--indexToPage(--index--)----------------------><
```

Returns the PROPSHEETPAGE *handle* for the specified page index.

Arguments:

The single argument is:

index [required]

The one-based page index.

Return value:

The PROPSHEETPAGE handle for the specified page.

Remarks:

The PROPSHEETPAGE handle is not the same as the window handle of the page. This method is added for completeness. In the current implementation of ooDialog, the returned handle will not be of much use to the programmer. However, future version of ooDialog may have enhancements that require a PROPSHEETPAGE handle.

Details

Raises syntax errors when incorrect usage is detected.

5.2.26. initDialog

```
>>--initDialog-----------------------------------><
```

Invoked by the ooDialog framework when the underlying property sheet dialog has first been created. This is the same point in the life-cycle of the property sheet dialog as the *initDialog* method is invoked for the normal *dialog* object.

Arguments:

There are no arguments to this method.

Return value:

Any return from this method is ignored.

Remarks:

This method is available for the programmer to over-ride for any purpose the programmer deems useful. However, since the operating system manages the property sheet dialog, this method will not be as useful as it is in the typical ooDialog dialog.

5.2.27. insertPage

```
>>--insertPage(--pageDlg--,--index--+---------------+--)--------><
                                    +-,-isExterior--+
```

Inserts a new page into the property sheet dialog at the specified index.

Arguments:

The arguments are:

pageDlg [required]

The Rexx dialog object to use as the page being inserted.

index

The one-based index where the page is to be inserted.

isExterior [optional]

Specifies if the inserted page is an exterior page of a wizard property sheet dialog. If the property sheet dialog is not a wizard, this argument is ignored.

Return value:

Returns true on success, otherwise false.

Remarks:

Syntax conditions are raised if the *page* argument is not correct. The argument must be a *PropertySheetPage* dialog object.

The pages after the insertion point are shifted to the right to accommodate the new page.

This method must not be invoked from within the event handling methods, *apply*, *killActive*, *reset*, or *setActive* of the *PropertySheetPage* class. However, you can add or remove pages during the *wizBack* or *wizNext* event handling methods, provided the correct index to specify the new page is returned from those event handlers.

The dialog object must not have already been *used* as a property sheet page. In addition to adding or inserting pages in a property sheet, pages can be removed from a property sheet. When the page is removed, the operating system destroys the underlying Windows dialog. Although the Rexx dialog object is still active, that object can not be reinserted into a property sheet as a page.

It is possible that the user never *visited* a page before it was removed. In this case the Rexx dialog object could be used to add or insert a new page at some later point, because the underlying Windows dialog would never have been created. The *wasActivated* attribute can be used to test for this condition. However, it is probably simplier to always instantiate a new Rexx dialog object for each page added to a property sheet.

The Windows operating system restricts the maximum number of pages that a property sheet can hold. The MAXPROPPAGES constant of the *PropertySheetDialog* reflects this number. If the programmer tries to insert or add a page past this number, a condition is raised.

Details

Raises syntax errors when incorrect usage is detected.

5.2.28. pageToIndex

```
>>--pageToIndex(--hPage--)-----------------------><
```

Takes the handle of a *indexToPage* and returns the one-based index of the page.

Arguments:

The single argument is:

hPage

The handle to the PROPSHEETPAGE whose index is needed.

Return value:

The one-based index of the page specified by *hPage*.

Remarks:

The PROPSHEETPAGE handle is not the same as the window handle of the page. This method is added for completeness. In the current implementation of ooDialog, the PROPSHEET handle is not of much use to the programmer. However, future version of ooDialog may have enhancements that require a PROPSHEETPAGE handle.

Details

Raises syntax errors when incorrect usage is detected.

5.2.29. popup

```
>>--popup----------------------------------------><
```

Begins the execution of a modeless property sheet dialog and returns immediately.

Arguments:

There are no arguments to this method.

Return value:

Returns true on success, otherwise false.

Remarks:

All **`PropertySheetDialog`** objects must be executed through either the *popup* or *execute* methods. However, a property sheet with the Aero Wizard style can not be executed as a modeless dialog. Do not use the *popup* method with an Aero Wizard.

5.2.30. popupAsChild

```
>>--popupAsChild(--parent--)---------------------><
```

The *popupAsChild* method starts a *modeless* property sheet dialog executing and returns immediately. It is very similar to the *popup* method, but has the additional functionality that it is closed automatically when the *parent* dialog is closed.

Arguments:

The single argument is:

parent [required]

Some other Rexx dialog object. A relationship is established with the *parent* dialog such that when the *parent* dialog is closed, this dialog is automatically closed also.

Return value:

The return is a Rexx **`Message`** object, (see the Open Object Rexx reference manual to learn more about the **`Message`** class.) The message object can be used, for instance, to check if the dialog has been closed by the user, to obtain the result value from the execution of the dialog, etc..

Remarks:

Note that the relationship between the dialog started with *popupAsChild* and the *parent* dialog is one-way. When the *parent* dialog is closed, the dialog started with the *popupAsChild* method is closed. But, when the dialog started with *popupAsChild* is closed, the *parent* dialog continues to execute as normal. It is not effected by the closing of the *child* dialog.

Example:

This example shows a property sheet dialog that is popped up from the parent dialog when a button is clicked:

```
::method onPopup unguarded

    -- Create the 5 dialog pages.
    p1 = .ListViewDlg~new("rc\PropertySheetDemo.rc", IDD_LISTVIEW_DLG)
    p2 = .TreeViewDlg~new("rc\PropertySheetDemo.rc", IDD_TREEVIEW_DLG)
    p3 = .ProgressBarDlg~new("rc\PropertySheetDemo.rc", IDD_PROGRESSBAR_DLG)
    p4 = .TrackBarDlg~new("rc\PropertySheetDemo.rc", IDD_TRACKBAR_DLG)
    p5 = .TabDlg~new("rc\PropertySheetDemo.rc", IDD_TAB_DLG)

    -- Create the PropertySheetDialog using an array of the 5 dialog pages.
    pages = .array~of(p1, p2, p3, p4, p5)
    title = "ooRexx Property Sheet with Controls"
    propDlg = .PropertySheetDemoDlg~new(pages, "NOAPPLYNOW", title)

    -- Show the property sheet.
    propDlg~popupAsChild(self)
```

5.2.31. pressButton

```
>>--pressButton(--button--)----------------------><
```

Pushes the specified property sheet button programmatically.

button [required]

Exactly one of the following keywords, case is not significant:

APPLYNOW	FINISH	OK
BACK	HELP	
CANCEL	NEXT	

APPLYNOW
: The ApplyNow button is pushed.

BACK
: The Back button is pushed.

Cancel
: The Cancel button is pushed.

Finish
: The Finish button is pushed.

Help
: The Help button is pushed.

Next
: The Next button is pushed.

OK
: The Ok button is pushed.

Return value:

Returns 0, always.

Details

Raises syntax errors when incorrect usage is detected.

5.2.32. querySiblings

```
>>--querySiblings(--arg1--,--arg2--)-------------><
```

The *querySiblings* method causes the underlying property sheet dialog to send the *queryFromSibling* notification, with the specified arguments to each page of the of the property sheet dialog.

Arguments:

The two arguments are sent in the QUERYFROMSIBLING notification exactly as specified. These arguments can be any Rexx object desired.

Return value:

Returns the nonzero value from a page in the property sheet, or zero if no page returns a nonzero value.

Remarks:

If a page returns a nonzero value, the property sheet does not send the message to subsequent pages.

The *querySiblings* method provides a way for the different page dialogs in a property sheet dialog to communicate with one another. The uses for this ability are certainly unlimited.

Example:

In this example, when one of the pages in a wizard becomes active it sends an empty table to each of the other pages in the property sheet combined with the command SELECTED. Each page then adds information to the table passed in with the command. The information is specific to the application:

```
::method setActive unguarded
  expose filmArray movieTheaters selectedMovies movieCombo
  use arg propSheet

  propSheet~setWizButtons("BACK NEXT")

  selected = .table~new
  selectedMovies~empty
  propSheet~querySiblings(selected, "SELECTED")
```

5.2.33. rebootSystem

```
>>--rebootSystem---------------------------------><
```

The *rebootSystem* method is invoked to notify the property sheet dialog that the system needs to be rebooted for changes to take effect.

Arguments:

There are no arguments for this method

Return value:

Returns 0 always.

Remarks:

An application should invoke the *rebootSystem* method only in response to the APPLY or KILLACTIVE notifications. Invoking the method causes the *execute* method to return the ID_PSREBOOTSYSTEM value, or for the *getResult* method to return *RebootSystem*. But only if the user clicks the OK button to close the property sheet. It is the application's responsibility to reboot the system.

5.2.34. recalcPageSizes

```
>>--recalcPageSizes------------------------------><
```

Causes a standard or wizard property sheet to recalculate the page size of all its pages. The *recalcPageSizes* method would normally be invoked after pages have been added or removed.

Arguments:

There are no arguments for this method.

Return value:

Returns true if successful, otherwise false.

Remarks:

When a property sheet is created, it is sized to fit its initial collection of pages. To maintain compatibility with previous versions of the common controls, property sheets and wizards do not automatically resize themselves when pages are subsequently added or removed. Applications would use the *recalcPageSizes* method after adding or removing pages to ensure that the property sheet is properly sized for its current collection of pages. **Note** however that the ooDialog framework automatically invokes this method during the *addPage*, *insertPage*, and *removePage* methods. Therefore the programmer will normally not need to explicitly invoke this method.

5.2.35. restartWindows

```
>>--restartWindows-------------------------------><
```

The *restartWindows* method is invoked to notify the property sheet dialog that Microsoft Windows needs to be restarted for changes to take effect.

Arguments:

This method has no arguments.

Return value:

Returns 0, always.

Remarks:

An application should invoke the *restartWindows* method only in response to the APPLY or KILLACTIVE notifications. Invoking the method causes the *execute* method to return the ID_PSRESTARTWINDOWS value, or for the *getResult* method to return *RestartWindows*. But only if the user clicks the OK button to close the property sheet. It is the application's responsibility to restart Windows.

5.2.36. removePage

```
>>--removePage(--+---------+--)------------------><
                 +--index--+
```

Removes a page from this property sheet.

Arguments:

The arguments are:

index [optional]
: The one-based index of the page to remove. If this argument is omitted then the last page is removed.

Return value:

Always returns 0.

Remarks:

This method must not be invoked from within the event handling methods, *apply*, *killActive*, *reset*, or *setActive* of the *PropertySheetPage* class. However, you can add or remove pages during the *wizBack* or *wizNext* event handling methods, provided the correct index to specify the new page is returned from those event handlers.

Details

Raises syntax errors when incorrect usage is detected.

5.2.37. setButtonText

```
>>--setButtonText(--button--,--text--)-----------><
```

Sets the text of the specified button in an Aero wizard. *Vista* or later only. This method does nothing if this property sheet is not an Aero wizard.

Arguments:

The arguments are:

button [required]

Exactly one of the following keyword, case is not significant. This specifies which property sheet button's text will be set.

BACK	FINISH
CANCEL	NEXT

BACK
: The Back button.

CANCEL
: The Cancel button.

FINISH
: The Finish button.

NEXT
: The Next button.

text [required]
: The text, the label, for the specified button.

Return value:

True if the property sheet is an Aero wizard and the label for the button was allocated, otherwise false.

Details

Raises syntax errors when incorrect usage is detected.

5.2.38. setCurSel

```
>>--setCurSel(--+----------+--+-----------------+--)----------><
                +-,-index--+  +--hPropSheetPage--+
```

Activates the specified page in a property sheet.

Arguments:

The arguments are:

index [optional]

The one-based index of the page to be activated. If this argument is omitted, then the *hPropSheetPage* argument can not be omitted.

hPropSheetPage [optional]

A *indexToPage handle* for the page to be activated If this argument is omitted, then the *index* argument can not be omitted.

Return value:

Returns true on success, othewise false.

Remarks:

The property sheet page to activate can be specified by either the page index, or the property sheet page handle, or both. If both are specified, the page index takes precedence.

Although both arguments are optional, they are optional individually. At least one of the arguments must be specified or a condition is raised. In addition, if the index argument is used to specify the page is not a valid index, a condition is raised.

Details

Raises syntax errors when incorrect usage is detected.

5.2.39. setCurSelByID

```
>>--setCurSelByID(--id--)-----------------------><
```

Activates the page specified by *id* in this property sheet.

Arguments:

The single argument is:

id

The page ID of the page to activate.

Return value:

Returns true on success, false on error.

Remarks:

The property sheet page ID can be obtained using the *indexToID* method. Do not confuse a page ID with a page handle, they are 2 separate things. The only way for the Rexx programmer to obtain the page ID is through *indexToID*.

Details

Raises syntax errors when incorrect usage is detected.

5.2.40. setFinishText

```
>>--setFinishText(--text--)----------------------><
```

Sets the text of the Finish button in a wizard, shows and enables the button, and hides the Next and Back buttons.

Arguments:

The single argument is:

text [required]

The text to display on the Finish button.

Return value:

Returns 0 always.

5.2.41. setHeaderSubTitle

```
>>--setHeaderSubTitle(--index--,--text--)--------><
```

Sets or resets the text for the header subtitle in a wizard property sheet dialog.

Arguments:

The arguments are:

index [required]

The one-based index of the page whose header subtitle is to be set.

text [required]

The text for the header subtitle.

Return value:

Always returns 0.

Remarks:

This method is not available for Aero-style wizards. If the page specified is the current page, it will immediately be repainted to display the new subtitle.

Details

Raises syntax errors when incorrect usage is detected.

5.2.42. setHeaderTitle

```
>>--setHeaderTitle(--index--,--text--)-----------><
```

Sets or resets the text for the header in a wizard property sheet dialog.

Arguments:

The arguments are:

index [required]

The one-based index of the page whose header text is to be set.

text [required]
    The text for the header.

Return value:
    Always returns 0.

Remarks:
    This method does not work for Aero-style wizards. If the page specified is the current page, it will immediately be repainted to display the new header.

Details
    Raises syntax errors when incorrect usage is detected.

5.2.43. setNextText

```
>>--setNextText(--text--)------------------------><
```

Sets the text of the Next button in an Aero wizard. *Vista* or later only. This method does nothing if this property sheet is not an Aero wizard.

Arguments:
    The single argument is:
    text [required]
        The text to display on the Next button.

Return value:
    This method always returns 0.

5.2.44. setTitle

```
>>--setTitle(--text--+---------------------+--)-----------------><
                     +-,-addPropertiesFor--+
```

Sets the title for a property sheet dialog. Testing has shown that this method does nothing for any property sheet that is a wizard.

Arguments:
    The arguments are:
    text [required]
        The text for the property sheet title.

    addPropertiesFor [optional]
        If true, the operating system includes the prefix *Properties for* with the specified title text. Otherwise, the prefix is not used. The default is false.

Return value:
    Returns 0, always.

Remarks:
    The MSDN documentation seems to indicate that this is valid for wizards, but experimentation shows it does not work for any wizard.

Details

Raises syntax errors when incorrect usage is detected.

5.2.45. setWizButtons

```
>>--setWizButtons(--+--------+--)----------------><
                    +--opts--+
```

Enables or disables the Back, Next, and Finish buttons in a wizard.

Arguments:

The single argument is.

opts [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant:

BACK	FINISH
DISABLEFINISH	NEXT

BACK

Enables the Back button. If this keyword is missing, the Back button is displayed as disabled.

DISABLEFINISH

Displays a disabled Finish button

FINISH

Displays an enabled Finish button.

NEXT

Enables the Next button. If this keyword is missing, the Next button is displayed as disabled.

Return value:

Returns true if the property sheet dialog is a wizard, otherwise returns false.

Remarks:

Wizards display either three or four buttons below each page. This method is used to specify which buttons are enabled. Wizards normally display Back, Cancel, and either a Next or Finish button.

Typically, the application enables only the Next button for the welcome page, enables the Next and Back buttons for the interior pages, and enables the Back and Finish buttons for the completion page. The Cancel button is always enabled. Normally, setting FINISH or DISABLEFINISH replaces the Next button with a Finish button. To changes this to display both Next and Finish buttons simultaneously, set the WIZARDHASFINISH keyword in the options when the *new* is instantiated. With that option, every page will then display all four buttons.

If this property sheet is not a Wizard, this method has no effect.

Example:

This example shows the *setActive* event handler in one of the interior page dialogs for a wizard property sheet dialog:

```
::method setActive unguarded
```

```
use arg propSheet

propSheet~setWizButtons("BACK NEXT")
return 0
```

5.2.46. showWizButtons

```
>>--showWizButtons(--+--------+--+---------------+--)---------><
                     +--opts--+  +-,-optsButtons--+
```

Shows or hides buttons in an Aero wizard. *Vista* or later only. This method does nothing if this property sheet is not an Aero wizard.

Arguments:

The arguments are:

opts [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. These specify which property sheet buttons are to be shown. If a button value is included in both this argument and *optsButton* then it is shown.

BACK	FINISH
CANCEL	NEXT

BACK

The Back button.

CANCEL

The Cancel button.

FINISH

The Finish button.

NEXT

The Next button.

opts [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. these specify which property sheet buttons are to be shown or hidden. If a button value appears in this parameter but not in dwFlag, it indicates that the button should be hidden.

BACK	FINISH
CANCEL	NEXT

BACK

The Back button.

CANCEL

The Cancel button.

FINISH

The Finish button.

NEXT

The Next button.

Return value:

Returns true if this is an Aero Wizard property sheet on Vista on later, otherwise false.

Remarks:

The order of *showWizButtons* and *setWizButtons* is important. This works:

```
propSheet~setWizButtons("NEXT")
propSheet~showWizButtons("NEXT", "BACK NEXT")
```

This does not work:

```
propSheet~showWizButtons("NEXT", "BACK NEXT")
propSheet~setWizButtons("NEXT")
```

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows the *setActive* event handler for the first page in an Aero wizard. It removes the Back button from the page, (since it is impossible to go back from the first page.)

```
::method setActive
  use arg propSheet

  propSheet~setWizButtons("NEXT")
  propSheet~showWizButtons("NEXT", "BACK NEXT")

  return 0
```

5.2.47. unchanged

```
>>--unchanged(--pspDlg--)------------------------><
```

Informs this property sheet that information in a page has reverted to the previously saved state.

Arguments:

The single argument is:

pspDlg

The Rexx property sheet page object whose data has been reverted.

Return value:

Returns zero, always.

Remarks:

The property sheet disables the Apply Now button if no other pages have registered changes with the property sheet. The MSDN documentation specifically states that this message does not apply to Aero wizards, it probably does nothing in all wizards.

Details

Raises syntax errors when incorrect usage is detected.

5.3. Property Sheet Pages

Each property sheet *PropertySheetDialog* must contain one or more property sheet *pages*.
To instantiate a **`PropertySheetDialog`** the programmer passes in an array of one or more **`PropertySheetPage`** objects. In ooDialog each property sheet page is represented by one of the concrete property sheet page dialogs that inherit from the **`PropertySheetPage`** mixin class.

To create a page for a **`PropertySheetDialog`** the programmer defines a dialog subclass using one of these three property sheet page dialogs:

- Subclass the *RcPSPDialog* when the page dialog template will be defined in a resource *script* file.
- Subclass the *ResPSPDialog* when the dialog *template* will be loaded from a compiled binary resource file.
- Subclass the *UserPSPDialog* when the dialog template will be defined using the *create...* methods.

All property sheet page dialogs are managed more by the operating system than is typical for an ooDialog dialog. The programmer does not *execute* the page dialog to create the *underlying* Windows dialog, or stop the dialog. The operating system creates and destroys the underlying page dialogs.

Each page has a corresponding icon and label. The underlying property sheet, that is the operating system, creates a tab for each page and displays the icon and label in the tab. Note that this does not apply to wizards. The minimum size for a property sheet page is 212 dialog units horizontally and 114 dialog units vertically. If a page dialog is smaller than this, the operating system will enlarge the page until it meets the minimum size.

Microsoft has three recommended sizes for pages and notes that using these recommended sizes will help ensure visual consistency between your application and other Microsoft Windows applications. The three sizes are basically *small*, *medium*, and *large*. The *pageOpts* argument to the *new* method of each of the three types of property sheet page dialogs provides the SMALL, MEDIUM, and LARGE keywords to automatically set the size of the property sheet page to one of the recommended sizes. The **`PropertySheetPage`** class provides *constant*) values for the width and height of the three recommended sizes for property sheet page dialogs.

5.3.1. PropertySheetPage Class

The **`PropertySheetPage`** class is a mixin class that contains the methods of a property sheet page dialog object. Other than the individual *new* methods for each of the three types of property sheet page dialogs, (Rc *new*, Res *new*, and User *new*,) all page methods come from the **`PropertySheetPage`** class. Of course, each property sheet page dialog has all the methods of the *dialog* object and the methods of its parent class, respectively the *RcDialog*, *ResDialog*, or *UserDialog*.

5.3.1.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with **`PropertySheetPage`** objects.

Table 5.5. PropertySheetPage Methods

Method	Description
Constant Methods	
PROP_SM_CXDLG	Width, in dialog units, of a small property sheet page. (Currently 212 dialog units.)
PROP_SM_CYDLG	Height, in dialog units, of a small property sheet page. (Currently 188 dialog units.)

Method	Description
PROP_MED_CXDLG	Width, in dialog units, of a medium property sheet page. (Currently 227 dialog units.)
PROP_MED_CYDLG	Height, in dialog units, of a medium property sheet page. (Currently 215 dialog units.)
PROP_LG_CXDLG	Width, in dialog units, of a large property sheet page. (Currently 252 dialog units.)
PROP_LG_CYDLG	Height, in dialog units, of a large property sheet page. (Currently 218 dialog units.)
PSNRET_NOERROR	A constant value that can be returned from a property sheet page event handler. The individual event handlers document the meaning of the PSNRET_xxx values for that handler.
PSNRET_INVALID	A constant value that can be returned from a property sheet page event handler. The individual event handlers document the meaning of the PSNRET_xxx values for that handler.
PSNRET_INVALID_NOCHANGEPAGE	A constant value that can be returned from a property sheet page event handler. The individual event handlers document the meaning of the PSNRET_xxx values for that handler.
PSNRET_MESSAGEHANDLED	A constant value that can be returned from a property sheet page event handler. The individual event handlers document the meaning of the PSNRET_xxx values for that handler.
Class Methods	
RcPSPDialog (RcPSPDialog)	Instantiates a new property sheet page whose dialog template is created from a resource *script*.
ResPSPDialog (ResPSPDialog)	Instantiates a new property sheet page whose dialog *template* comes from a compiled binary resource file.
UserPSPDialog (UserPSPDialog)	Instantiates a new property sheet page whose dialog template is created using the *create...* methods of the **`UserDialog`** class.
Attributes	
cx	Reflects the initial width of the property page dialog in dialog units.
cy	Reflects the initial height of the property page dialog in dialog units.
headerSubtitle	Reflects the text for the subtitle of a non-Aero wizard.
headerTitle	Reflects the text that will be displayed as the title in the header area of the page in a wizard.
pageID	Reflects the value of the page ID, which identifies the page to the *underlying* property sheet dialog.
pageNumber	Reflects the page number, or index, of this property sheet page.
pageTitle	Reflects the title to use for this property sheet page.
propSheet	Reflects the Rexx *PropertySheetDialog* object that this page belongs to.
resources	The *resources* attribute is set to a *ResourceImage* object containing resources used by this property sheet page.
tabIcon	Reflects the icon to use as the icon in the tab of this property sheet page.

Method	Description
wasActivated	Reflects whether the *underlying* property sheet page was created.
wantAccelerators	The value of this attribute determines if this page should receive the TRANSLATEACCELERATOR event notification.
wantGetObject	The value of this attribute determines if this page should receive the GETOBJECT event notification message.
Instance Methods	
apply	Notifies every page in the property sheet that the user has clicked the OK, Close, or Apply button and wants all changes to take effect.
getObject	Sent by a property sheet dialog to request a drop target object when the cursor passes over one of the tab control's buttons.
help	Notifies a page that the user has clicked the Help button.
killActive	Notifies a page that it is about to lose activation either because another page is being activated or the user has clicked the OK button.
pageCreate	Notifies the Rexx page dialog that the *underlying* page dialog is about to be created.
queryCancel	Notifies a property sheet page that the user has canceled the property sheet dialog.
queryFromSibling	Notifies the page of a *querySiblings*messge.
queryInitialFocus	Provides a property sheet page an opportunity to specify which dialog box control should receive the initial focus.
reset	Notifies a page that the property sheet is about to be destroyed.
setActive	Notifies a page that it is about to be activated.
setSize	Used to set both the *cx* and *cy* attributes at one time.
translateAccelerator	Notifies a property sheet page that a keyboard message has been received. It provides the page an opportunity to do private keyboard accelerator translation.
wizBack	Notifies a page that the user has clicked the Back button in a wizard.
wizFinish	Notifies a page that the user has clicked the Finish button in a wizard.
wizNext	Notifies a page that the user has clicked the Next button in a wizard.
validate	Provides a place to validate the data entered by the user in a page during the *apply* notification.
validatePage	Provides a place to validate the data entered by the user in a page during the *killActive* notification

5.3.1.2. Page Event Handlers

The process for connecting *event* handlers for the property sheet page dialogs takes a slightly different approach than that of other dialog classes in the ooDialog framework. The property sheet page class provides a default event handler for all property sheet page specific events.

Rather than connecting a property sheet page event to a specified method in the Rexx dialog, the programmer over-rides the default handler in the dialog. This is similar to how the *ok* and *cancel* events are handled in the *dialog* object.

In the property sheet page dialogs, most of the methods are actually default event handlers. To act on an event in a manner different from the default, the programmer over-rides the default handler and provides an implementation that handles the event as needed by the application. For example, to remove the back button from the first page of an Aero wizard, the programmer would over-ride the *setActive* method in the page dialog for the first page of the property sheet dialog. In the new implementation code for *setActive*, the programmer would invoke the *showWizButtons* method of the property sheet dialog and specify that the Back button was hidden. This same general technique is used for all property sheet page events.

5.3.1.3. cx (Attribute)

```
>>--cx-------------------------------------------><

>>--cx-=-width-----------------------------------><
```

Reflects the initial width of the property page dialog in dialog units.

cx get:

The value of the *cx* attribute is the width, in dialog *dialog unit*s, of the page dialog.

cx set:

When the page dialog is first instantiated, *cx* is set to one of the 3 recommended page widths though the use of the SMALL, MEDIUM, or LARGE keywords of the *pageOpts* argument to the *new* method of the page dialog. Since the MEDIUM keyword is the default if the SMALL or LARGE keywords are not used, *cx* is always set to a value. The programmer can reset the *cx* value if desired.

Remarks:

Setting *cx* is really only of use for a *UserPSPDialog*. And, indeed, for a **`UserPSPDialog`** the programmer must set this attribute to obtain a dialog with a different width than one of the 3 recommended widths. With a *RcPSPDialog*, *cx* is over-written with the value in the resource script, and for a *ResPSPDialog*, the width is determined when the resource is compiled and can not changed.

With a **`UserPSPDialog`**, the underlying page dialog is created with the width specified by the *cx* attribute. So, to have a different width than one of the recommended widths, the programmer must set this attribute before the dialog template is created. Note that the *setSize* method can be used to set both the *cx* and *cy* attributes at one time.

Example:

This example shows an **`UserPSPDialog`** being instantiated for use as one of the pages of a *PropertySheetDialog* sheet dialog. The size of the page dialog is set using the *cx* and *cy* attributes;

```
  p3 = .DaysDlg~new( , , "", "Days" )
  p3~cx = 267
  p3~cy = 143
  p3~headerTitle = "Choose a day to watch the movie(s)."
  p3~headerSubtitle = "Use the drop down list to switch movies.  Pick the day most
suitable for the movie."
  ...
```

5.3.1.4. cy (Attribute)

```
>>--cy-------------------------------------------><

>>--cy-=-height----------------------------------><
```

Reflects the initial height of the property page dialog in dialog units.

cy get:

The value of the *cy* attribute is the height, in dialog *dialog unit*s, of the page dialog.

cy set:

When the page dialog is first instantiated, *cy* is set to one of the 3 recommended page heights though the use of the SMALL, MEDIUM, or LARGE keywords of the *pageOpts* argument to the *new* method of the page dialog. Since the MEDIUM keyword is the default if the SMALL or LARGE keywords are not used, *cy* is always set to a value. The programmer can reset the *cy* value if desired.

Remarks:

Setting *cy* is really only of use for a *UserPSPDialog*. And, indeed, for a **`UserPSPDialog`** the programmer must set this attribute to obtain a dialog with a different height than one of the 3 recommended heights. With a *RcPSPDialog*, *cy* is over-written with the value in the resource script, and for a *ResPSPDialog*, the height is determined when the resource is compiled and can not changed.

With a **`UserPSPDialog`**, the underlying page dialog is created with the height specified by the *cy* attribute. So, to have a different height than one of the recommended heights, the programmer must set this attribute before the dialog template is created. Note that the *setSize* method can be used to set both the *cy* and *cx* attributes at one time.

Example:

This example shows an **`UserPSPDialog`** being instantiated for use as one of the pages of a *PropertySheetDialog* sheet dialog. The size of the page dialog is set using the *cx* and *cy* attributes;

```
  p3 = .DaysDlg~new( , , "", "Days" )
  p3~cx = 267
  p3~cy = 143
  p3~headerTitle = "Choose a day to watch the movie(s)."
  p3~headerSubtitle = "Use the drop down list to switch movies.  Pick the day most
suitable for the movie."
  ...
```

5.3.1.5. headerSubtitle (Attribute)

```
>>--headerSubtitle-------------------------------><

>>--headerSubtitle-=-text------------------------><
```

Reflects the text for the subtitle of a non-Aero wizard.

headerSubtitle get:

The value of the *headerSubtitle* attribute is the text a wizard property sheet page will use for the subtitle text in the header area of a wizard. If the programmer has not assigned a value to this attribute, its value is the **`.nil`** object.

headerSubtitle set:

To display text for the subtitle in a wizard dialog page, the programmer can set the *headerSubtitle* attribute to the text to be displayed. This text is used when the underlying page is first created. Setting the attribute after the page has been created has no effect. Likewise, setting this attribute to any text has no effect if the page is not in a wizard dialog. Aero-style wizards do not use subtitle text.

Remarks:

For non-aero wizards, the subtitle text can also be set dynamically by using the *setHeaderSubTitle* method of the *PropertySheetDialog*. For instance, setting the subtitle text could be done in the *setActive* event handler when the page is about to become active but before it is actually shown on the screen.

5.3.1.6. headerTitle (Attribute)

```
>>--headerTitle----------------------------------><

>>--headerTitle-=-text---------------------------><
```

Reflects the text that will be displayed as the title in the header area of the page in a wizard.

headerTitle get:

The value of the *headerTitle* attribute is the text a wizard property sheet page will use for the title text in the header area of a wizard. If the programmer has not assigned a value to this attribute, its value is the **`.nil`** object.

headerTitle set:

To display text for the header title in a wizard dialog page, the programmer can set the *headerTitle* attribute to the text to be displayed. This text is used when the underlying page is first created. Setting the attribute after the page has been created has no effect. Likewise, setting this attribute to any text has no effect if the page is a page in a non-wizard dialog.

Remarks:

For non-aero wizards, the header title text can also be set dynamically by using the *setHeaderTitle* method of the *PropertySheetDialog*. For instance, setting the title text could be done in the *setActive* event handler when the page is about to become active but before it is actually shown on the screen.

5.3.1.7. pageID (Attribute)

```
>>--pageID---------------------------------------><

>>--pageID-=-val---------------------------------><
```

Reflects the value of the page ID, which identifies the page to the *underlying* property sheet dialog. The ID is essentially an opaque value to the Rexx programmer.

pageID get:

The value of the *pageID* attribute is a value that uniquely identifies this page to the underlying property sheet dialog that this page belongs to. In Rexx, the value is a **`Pointer`** object. The value of the **`Pointer`** has no meaning to the programmer.

pageID set:

The Rexx programmer can not set this value it is set internally by the ooDialog framework.

Remarks:

Note that the page *ID* is different than the page *index*. The *index* is the page number of the page, page 1, page 2, etc.. A page ID can be returned from several of the page event handlers, such as *setActive*, *wizBack*, and *wizNext* to direct the property sheet dialog to display the page with that ID. The *indexToID* method of the *PropertySheetDialog* class can be used to translate the page number to that page's ID.

5.3.1.8. pageNumber (Attribute)

```
>>--pageNumber-----------------------------------><

>>--pageNumber-=-val-----------------------------><
```

Reflects the page number, or index, of this property sheet page.

pageNumber get:

The property sheet page numbers are one-based and the *pageNumber* attribute is the current page number in the property sheet dialog for this property sheet page.

pageNumber set:

The programmer can not set the *pageNumber* attribute, it is set correctly by the ooDialog framework.

5.3.1.9. pageTitle (Attribute)

```
>>--pageTitle------------------------------------><

>>--pageTitle-=-text-----------------------------><
```

Reflects the title to use for this property sheet page.

pageTitle get:

The value of the *pageTitle* attribute is the text used for the label of the tab for this property sheet page. If the programmer has not assigned a value to this attribute, its value is the **`.nil`** object.

pageTitle set:

The programmer can set the *pageTitle* attribute to the text to be used for the label of the tab in a property sheet dialog for this page. This text is used when the underlying page is first created. Setting the attribute after the page has been created has no effect.

Remarks:

Normally, the property sheet dialog sets the title of a property sheet page to the caption in the dialog template for the page. The title of a property sheet page is used as the label of the tab for that page. If the dialog template does not have a caption for the dialog, or if the programmer wants a label different from the caption in the dialog template, then the *pageTitle* attribute should be set to the desired label for the tab.

In addition to setting the *pageTitle* attribute, the USETITLE keyword needs to be included in the *pageOpts* argument to the *new* method of the page. However, in general, the ooDialog framework will set the keyword correctly if / when the *pageTitle* attrbiute is set, and the Rexx programmer does not need to worry about it. There is one *exception* to this. For the *RcPSPDialog* page object, the title for the page will be overwritten when the resource script is parsed. To prevent this, for **`RcPSPDialog`** pages, the USETITLE keyword must be used in the *pageOpts* argument.

In a *UserPSPDialog*, the *pageTitle* attribute can also be set by using the *title* argument in the *new* method of the class.

Example:

This example uses the dialog caption as set in the resource script file for the labels of the tabs, except for page 5. For page 5 the label is changed to *ooRexx Tab Control* by setting the *pageTitle* attribute. Notice the use of the USETITLE keyword in the *pageOpts* argument to the *new* method for page 5:

```
  -- Create the 5 dialog pages.
  p1 = .ListViewDlg~new("rc\PropertySheetDemo.rc", IDD_LISTVIEW_DLG)
  p2 = .TreeViewDlg~new("rc\PropertySheetDemo.rc", IDD_TREEVIEW_DLG)
  p3 = .ProgressBarDlg~new("rc\PropertySheetDemo.rc", IDD_PROGRESSBAR_DLG)
  p4 = .TrackBarDlg~new("rc\PropertySheetDemo.rc", IDD_TRACKBAR_DLG)
  p5 = .TabDlg~new("rc\PropertySheetDemo.rc", IDD_TAB_DLG, , , "USETITLE")

  p5~pageTitle = "ooRexx Tab Control"
```

5.3.1.10. propSheet (Attribute)

```
>>--propSheet------------------------------------><

>>--propSheet-=-propSheetObj---------------------><
```

Reflects the Rexx *PropertySheetDialog* object that this page belongs to.

propSheet get:

Returns the Rexx property sheet dialog that owns this property sheet page.

propSheet set:

The Rexx programmer can not set the value of this attribute. It is set internally by the ooDialog framework.

Remarks:

One of the arguments sent to the event handler methods of a property sheet page is the Rexx property sheet dialog object. That object sent as an argument and the *propSheet* attribute are the same Rexx object. Within an event handler the programmer can use either object to access the property sheet dialog, whichever seems more convenient.

Example:

This example uses the *propSheet* attribute to invoke the *querySiblings* method of the property sheet dialog that owns the page:

```
::method initDialog
  expose filmArray movieTheaters selectedMovies movieCombo

  filmArray = .array~new(20)
  self~propSheet~querySiblings(filmArray, "FILMS")

  ...
```

5.3.1.11. resources (Attribute)

```
>>--resources------------------------------------><

>>--resources-=-image----------------------------><
```

The *resources* attribute is set to a *ResourceImage* object containing resources used by this property sheet page.

resources get:

Returns the **`ResourceImage`** object if this attribute has been set by the programmer or the **`.nil`** object if the attrbute has not been set.

resources set:

The *resources* attribute is set to the source of some resources used by the property sheet page dialog. The object assigned to the *resources* attribute must be a *ResourceImage* object.

Remarks:

Resources used by a page dialog can be taken from a compiled binary executable module. In Windows, executable modules are **`*.exe`** and **`*.dll`** files. In the ooDialog framework, access to a compiled binary module is provided by the *ResourceImage* class.

Typically, the resources used by the property sheet page would include the tab icon and the strings used for the header title, header subtitle, page title, etc.. However, the current implementation does not yet support retrieving string resources from a **`ResourceImage`**. Therefore, at this point, it is really only the tab icon for this page that would be contained in the *resources* attribute. Future versions of ooDialog will likely be enhanced to work with string resources in a resource image.

To supply the tab icon for this page from a resource image, the programmer would set the *resources* attribute to a resource image containing the icon and set the *tabIcon* attribute to the resource ID of the icon in the resource image.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example creates a resource image from the **`ooDialog.dll`** file which has 4 icon images within the DLL. It assigns the resource image to the *resources* attribute so that it can use the icon images as the icons for the tabs of the pages:

```
p1 = .ListViewDlg~new("rc\PropertySheetDemo.rc", IDD_LISTVIEW_DLG)
p2 = .TreeViewDlg~new("rc\PropertySheetDemo.rc", IDD_TREEVIEW_DLG)
p3 = .ProgressBarDlg~new("rc\PropertySheetDemo.rc", IDD_PROGRESSBAR_DLG)
p4 = .TrackBarDlg~new("rc\PropertySheetDemo.rc", IDD_TRACKBAR_DLG)

resourceImage = .ResourceImage~new("oodialog.dll", p1)

p1~resources = resourceImage
p1~tabIcon = IDI_DLG_OODIALOG

p2~resources = resourceImage
p2~tabIcon = IDI_DLG_APPICON

p3~resources = resourceImage
p3~tabIcon = IDI_DLG_APPICON2
```

```
    p4~resources = resourceImage
    p4~tabIcon = IDI_DLG_OOREXX
```

5.3.1.12. tabIcon (Attribute)

```
>>--tabIcon--------------------------------------><

>>--tabIcon-=-icon-------------------------------><
```

Reflects the icon to use as the icon in the tab of this property sheet page.

tabIcon get:

The value of the *tabIcon* attribute if the programmer has set that value. Otherwise, the value is the **`.nil`** object.

tabIcon set:

This attribute can either be set to a stand alone icon *Image* object or to the resource ID of the icon if the icon image is contained in a compiled resource file. If the attribute is set to a resource ID, then the *resources* attribute must be set to a *ResourceImage* object that contains the icon image. If the attribute is set to an **`Image`** object, then the image has to be an actual icon image. It can not be a bitmap or cursor.

When a resource ID is used it may be numeric or *symbolic*.

Remarks:

The tab icons for all pages in a *PropertySheetDialog* sheet dialog can also be set through the *imageList* attribute. If both the *imageList* attribute and the *tabIcon* attribute are set, the icon for the tab is taken from the *imageList* attribute.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example creates a resource image from the ooDialog.dll file which has 4 icon images within the resource section of the DLL. It then assigns an icon from the resource image to the *tabIcon* attribute of each page dialog:

```
    p1 = .ListViewDlg~new("rc\PropertySheetDemo.rc", IDD_LISTVIEW_DLG)
    p2 = .TreeViewDlg~new("rc\PropertySheetDemo.rc", IDD_TREEVIEW_DLG)
    p3 = .ProgressBarDlg~new("rc\PropertySheetDemo.rc", IDD_PROGRESSBAR_DLG)
    p4 = .TrackBarDlg~new("rc\PropertySheetDemo.rc", IDD_TRACKBAR_DLG)

    resourceImage = .ResourceImage~new("oodialog.dll", p1)

    p1~resources = resourceImage
    p1~tabIcon = IDI_DLG_OODIALOG

    p2~resources = resourceImage
    p2~tabIcon = IDI_DLG_APPICON

    p3~resources = resourceImage
    p3~tabIcon = IDI_DLG_APPICON2

    p4~resources = resourceImage
    p4~tabIcon = IDI_DLG_OOREXX
```

5.3.1.13. wasActivated (Attribute)

```
>>--wasActivated---------------------------------><

>>--wasActivated-=-trueFalse---------------------><
```

Reflects whether the *underlying* property sheet page was created.

wasActivated get:

The value of this attribute is true if the underlying page was created, otherwise it is false.

wasActivated set:

The programmer can not set the value of this attribute, it is set internally by the ooDialog framework.

Remarks:

The property *PropertySheetDialog* dialog creates the property sheet page dialogs as they are needed, usually not until the page is visited. This attribute allows the Rexx programmer to determine whether or not the underlying dialog was ever created.

5.3.1.14. wantAccelerators (Attribute)

```
>>--wantAccelerators-----------------------------><

>>--wantAccelerators-=-trueFalse-----------------><
```

The value of this attribute determines if this page should receive the TRANSLATEACCELERATOR event notification. The notification is passed on to the page dialog by invoking its *translateAccelerator* method.

wantAccelerators get:

If the *wantAccelerators* attrbiute is true, the *translateAccelerator* method will be invoked for each TRANSLATEACCELERATOR event notification. By default this attribute is false and TRANSLATEACCELERATOR notifications are ignored.

wantAccelerators set:

By default, the ooDialog framework sets the *wantAccelerators* attribute to false. If the programmer wants to process the TRANSLATEACCELERATOR notification then this attribute value can be set to true.

Remarks:

Setting the *wantAccelerators* attribute to true won't by itself give the programmer notification of the translate accelerator event because the default *translateAccelerator* event handler just returns 0. The programmer must also over-ride the *translateAccelerator* method in her subclass and process the notification in the over-ride.

5.3.1.15. wantGetObject (Attribute)

```
>>--wantGetObject--------------------------------><

>>--wantGetObject=trueFalse----------------------><
```

The value of this attribute determines if this page should receive the GETOBJECT event notification message.

wantGetObject get:

Returns the value of the *wantGetObject* attribute. By default the value is false.

wantGetObject set:

Set the value of *wantGetObject* to true to have the GETOBJECT event notification passed on to this page.

Remarks:

This attribute and the *getObject* event handler are provided for completeness. The ooDialog framework does not currently have a way to allow the programmer to handle the get object event. The *wantGetObject* attribute is set to false by default. Changing it to true will not be of any use int the current ooDialog implementation.. Future versions of ooDialog may be enhanced to take advantage of this attribute.

5.3.1.16. apply

```
>>--apply(--isOkBtn-,-pageNum-,-rexxPage-,-propSheet--)---------><
```

Notifies every page in the property sheet that the user has clicked the OK, Close, or Apply button and wants all changes to take effect.

Arguments:

The arguments are:

isOkBtn

This argument is true if the user clicked the OK button. It is also set to true if the *cancelToClose* method had inovked on the property sheet and the user clicked the Close button. It is set to false if the user clicked the Apply button.

pageNum

This argument is set to the page number of the page that had the focus when the user clicked the button.

rexxPage

This argument is set to the Rexx property sheet page object that had the focus when the user clicked the button.

propSheet

This argument is set to the Rexx property sheet dialog object that owns this property sheet page.

Return value:

One of the following values must be returned:

PSNRET_NOERROR

Return the PSNRET_NOERROR *constant*) to indicate that the changes made to this page are valid and have been applied. If all pages return PSNRET_NOERROR, the property sheet dialog can be destroyed.

PSNRET_INVALID

Return the PSNRET_INVALID *constant*) to prevent the property sheet dialog from being destroyed. The focus will be returned to this page.

PSNRET_INVALID_NOCHANGEPAGE

Return the PSNRET_INVALID_NOCHANGEPAGE *constant*) to prevent the property sheet dialog from being destroyed. The focus will be returned to the page that had the focus when the button was pressed.

Remarks:

The default *apply* implementation returns PSNRET_NOERROR. When over-riding this method the programmer must return one of the three listed return values.

Over-riding apply():

The *apply* event handler can be over-ridden by the Rexx programmer, but there are a few additional details the programmer needs to be aware of. First, the default implementation simply calls the *validate* method and returns the value that *validate* returns. Because of that, an appropriate strategy would be to over-ride *validate* and leave the default implementation of *apply* as is.

Second, when the return from *apply* is to be PSNRET_NOERROR, the programmer should always invoke the private *applyNoError* method as the last step. Failure to invoke *applyNoError* will likely introduce inconsistencies in the behaviour of the page dialog as compared to the behaviour of other ooDialog dialogs.

Details

Raises syntax errors when incorrect usage is detected.

5.3.1.17. applyNoError

```
>>--applyNoError(--propSheet--)------------------><
```

The *applyNoError* method is a **private** method that is used by the ooDialog framework to keep the property sheet page dialogs behaviour consistent with the behaviour of other ooDialog dialogs. It is intended to be *invoked* by the Rexx programmer, but **not** over-ridden by the programmer.

The *applyNoError* should only be invoked from the *apply* method as the last step when PSN_NOERROR is going to be returned from *apply*. If the *apply* event handler is over-ridden by the programmer, the *applyNoError* methods should *always* be invoked as the final step when the return from *apply* will be PSN_NOERROR and it should *never* be invoked when the return is to be PSNRET_INVALID or PSNRET_PSNRET_INVALID_NOCHANGEPAGE.

The first and only argument to *applyNoError* should be the same *propSheet* argument that is passed to *apply*. The return from *applyNoError* is always PSNRET_NOERROR. The typical over-ride of the *apply* method would always look similar to this:

```
::method apply unguarded
  use arg isOkBtn, pageNum, rexxPage, propSheet

  ...
  -- Do all validation

  ...

  -- okay, everything is correct, return PSNRET_NOERROR

  return self~applyNoError(propSheet)
```

5.3.1.18. getObject

```
>>--getObject(--propSheet--)---------------------><
```

Sent by a property sheet dialog to request a drop target object when the cursor passes over one of the tab control's buttons. This event handler is provided for completeness. The method is currently a no-op. The ooDialog framework currently has no way for the programmer to respond to this notification. Future versions of ooDialog may be enhanced to allow an application to respond to this notification.

Arguments:

The single argument is:

propSheet

This argument is set to the Rexx property sheet dialog object that owns this property sheet page.

Return value:

Any return from *getObject* is ignored. The default implementation returns 0.

Remarks:

By default, the GETOBJECT notification is completely ignored. This is controlled by the *wantGetObject* attribute. If that attribute is set to true, the *getObject* event handler will be invoked.

5.3.1.19. help

```
>>--help(--propSheet--)--------------------------><
```

Notifies a page that the user has clicked the Help button.

Arguments:

The single argument is:

propSheet

This argument is set to the Rexx property sheet dialog object that owns this property sheet page.

Return value:

The default implementation of the *help* event handler returns 0. When the programmer over-rides this method, the method must return a value, but the value itself is ignored.

Remarks:

If the application provides a help button, then the application should display help if this notification is received.

Details

Raises syntax errors when incorrect usage is detected.

5.3.1.20. killActive

```
>>--killActive(--propSheet--)--------------------><
```

Notifies a page that it is about to lose activation either because another page is being activated or the user has clicked the OK button.

Arguments:

The single argument is:

propSheet

This argument is set to the Rexx property sheet dialog object that owns this property sheet page.

Return value:

The Rexx programmer must return **`.true`** to allow the page to lose activation and **`.false`** to prevent the page from losing the activation. The default implementation of this event handler returns *validatePage*. The default implementation of *validatePage* returns **`.true`**.

Remarks:

An application handles this notification to validate the information the user has entered. If the programmer returns **`.false`** to prevent the page from losing activation, it should display a message box to explain the problem to the user.

The default implementation invokes the *validatePage* method. Since the default implementation of *validatePage* returns **`.true`**, the default *killActive* returns **`.true`**. Rather than over-ride *killActive*, the programmer could over-ride the *validatePage* method and do page validation in the over-ride. Returning **`.false`** from the over-ride will cause the default *killActive* to return **`.false`**.

A common strategy might be to over-ride *validatePage*, do all page validation in *validatePage*, return **`.true`** or **`.false`** from *validatePage* depending on the outcome of the validation, and leave the default implementation of *killActive* alone.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example checks that the user has entered a last name in the form before leaving a page. If the last name field was left blank, the user is not allowed to leave the page:

```
::method killActive unguarded
  expose editLName
  use arg propSheet

  if editLName~getText~strip~length == 0 then do
    msg   = "The last name field must be filled in."
    title = "Acme Employment Application"

    ret = MessageDialog(msg, self~hwnd, title, "OK", "WARNING")

    editLName~assignFocus
    return .false
  end

  return .true
```

5.3.1.21. pageCreate

```
>>--pageCreate(--propSheet--)--------------------><
```

Notifies the Rexx page dialog that the *underlying* page dialog is about to be created. This notification is sent before the underlying page exists.

Arguments:

The single argument is:

propSheet

This argument is set to the Rexx property sheet dialog object that owns this property sheet page.

Return value:

Return **`.true`** to allow the page to be created. Return **`.false`** to *prevent* the underlying dialog page from being created. The default implmentation returns **`.true`**.

Remarks:

This event handler is provided for completeness. It is not readily apparent how useful handling the event would be.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example comes from some code that experiments with property sheet dialogs. It over-rides the default implementation of *pageCreate* to print a message when the *pageCreate* is invoked. It returns **`.false`** to prevent the page from being created. The experimentation showed that if the current tab selection is not set to another page, the appearance of the dialog was a little raggedy:

```
::method pageCreate unguarded
  use arg propSheet
  say 'In pageCreate(), propSheet:' propSheet
  reply .false
  propSheet~setCurSel(3)
```

5.3.1.22. queryCancel

```
>>--queryCancel(--propSheet--)-------------------><
```

Notifies a property sheet page that the user has canceled the property sheet dialog.

Arguments:

The single argument is:

propSheet

This argument is set to the Rexx property sheet dialog object that owns this property sheet page.

Return value:

The Rexx programmer must return **`.false`** to disallow the dialog from closing and **`.true`** to allow the dialog to close. The default implementation returns **`.true`**.

Remarks:

This event notification is typically sent when a user clicks the Cancel button. It is also sent when a user clicks the X button in the property sheet's upper right hand corner or presses the ESCAPE key. A property sheet page could handle this notification, for example, to ask the user to verify the cancel operation.

Details

Raises syntax errors when incorrect usage is detected.

5.3.1.23. queryFromSibling

```
>>--queryFromSibling(--arg1--,--arg2--,--propSheet--)-----------><
```

Notifies the page of a *querySiblings* messge.

Arguments:

The arguments are:

arg1
: The first application defined value. This can be any Rexx object.

arg2
: The second application defined value. This can be any Rexx object.

propSheet
: This argument is set to the Rexx property sheet dialog object that owns this property sheet page.

Return value:

Return a whole number. If the returned number is 0, the property sheet *PropertySheetDialog* sends the QUERYSIBLINGS message on to the next page in the property sheet. If the number is not 0, then the property sheet dialog does not send the message on the next page and returns its value from the *querySiblings* method. The default implementation returns 0.

Remarks:

The *querySiblings* method provides a way for the pages of a property sheet to communicate with each other. The *querySiblings* method is invoked with two values. The property sheet dialog then notifies each page, in order, through the *queryFromSibling* event handler. The two values passed into the *querySiblings* method are passed on unchanged to each page. If a page returns non-zero, then the property sheet does not notify any of the subsequent pages and returns from the *querySiblings* method. If all pages return 0, then the property sheet dialog returns 0 from the *querySiblings* method.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example comes from a movie ticket purchasing wizard. On the last page the application needs to know what movies the user is buying tickets for. In the last page, he application invokes *querySiblings* and all the other pages fill in the details of the tickets being purchased:

```
-- snippet code from the last page in a property sheet dialog

info = .table~new
movieCombo~deleteAll

ret = propSheet~querySiblings(info, "DATA")

do movie over info
  movieCombo~add(movie)
end
```

```
  -- snippet of code from the first page in the property sheet dialog

  when arg2 == "DATA" then do
    -- arg1 is a table whose indexes are the films.  Each item is a directory,
    -- and the indexes are theater, date, and time.  The other pages fill in the
    -- directory.  We just make sure only the selected movies are in the
    -- table.
    arg1~empty
    currentSelectedMovies = self~getSelectedMovies

    do movie over currentSelectedMovies
      d = .directory~new
      arg1[movie] = d
    end
  end
```

5.3.1.24. queryInitialFocus

```
>>--queryInitialFocus(--idDefault--,--propSheet--)--------------><
```

Provides a property sheet page an opportunity to specify which dialog box control should receive the initial focus.

Arguments:

The arguments are:

idDefault

The resource ID of the dialog control that will receive the focus by default

propSheet

This argument is set to the Rexx property sheet dialog object that owns this property sheet page.

Return value:

The Rexx programmer returns 0 to set the focus to the default control and returns the dialog control resource ID to set that focus to that control. The ID can be numeric or symbolic. The default implmentation returns 0.

Remarks:

The application must not invoke any ooDialog framework methods to set the focus while handling this notification. Instead, return the resource ID of the control that should receive focus, and the property sheet manager will handle the focus change.

Details

Raises syntax errors when incorrect usage is detected.

Example:

In this example the *queryInitialFocus* event handler is used to change the initial focus from the top check box to the third check box, using the symbolic ID of the third check box:

```
::method queryInitialFocus
  use arg id, propSheet

  return IDC_CK_NO_ADMIN
```

5.3.1.25. reset

```
>>--reset(--isCancelButton--,--propSheet--)------><
```

Notifies a page that the property sheet is about to be destroyed.

Arguments:

The arguments are:

isCancelButton

This argument is **`.true`** if the user clicked the Cancel button . It will be **`.false`** if the user clicked the X button in the upper-right corner of the property sheet.

propSheet

This argument is set to the Rexx property sheet dialog object that owns this property sheet page.

Return value:

The Rexx programmer must return a value from this event handler, but the value of the return has no meaning. The default implementation returns 0.

Remarks:

This notification is only sent to the page that has the current focus. It gives the page the opportunity to do some final clean up.

Details

Raises syntax errors when incorrect usage is detected.

5.3.1.26. setActive

```
>>--setActive(--propSheet--)---------------------><
```

Notifies a page that it is about to be activated.

Arguments:

The single argument is:

propSheet

This argument is set to the Rexx property sheet dialog object that owns this property sheet page.

Return value:

The Rexx programmer returns 0 to allow the page to become active, -1 to prevent a page change, and the page *pageNumber* or page *pageID*, to go to that specific page. The default implementation returns 0.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows the *setActive* method from the first page in an Aero wizard. It uses the notification to enable the Next button and to remove the Back and Finish buttons. (The user can not go *back* from the first page and *finishing* is not possible at that point.)

```
::method setActive
  use arg propSheet

  propSheet~setWizButtons("NEXT")
  propSheet~showWizButtons("NEXT", "BACK FINISH NEXT")

  return 0
```

5.3.1.27. setSize

```
Form 1:

>>--setSize(--sizeObj--)-------------------------><

Form 2:

>>--setSize(--width--,--height--)----------------><

Generic form:

>>--setSize(--sizeSpecifier--)-------------------><
```

The *setSize* method can be used to set both the *cx* and *cy* attributes at one time.

Arguments:

The arguments are the width and height values, in dialog units, for the dialog template of this page. As noted in the syntax diagram, the width and height can either be supplied through a single *Size* object, or by specifying the width and height separately.

Return value:

Returns 0, always.

Remarks:

This is a convenience method to set both the *cx* and *cy* attributes at one time. The documentation on the two attributes is relevant here. This is one of the few **`PropertySheetPage`** methods that is not an event handler.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This snippet of code shows how the *setSize* method can be used to set all the pages of a wizard dialog to the proper size:

```
  -- First we create the dialogs for each page, and set any of the attributes
  -- we need to.  We want each dialog to be the same size.
  outPageSize = .Size~new(267, 193)
  inPageSize = .Size~new(267, 143)

  intro = .IntroDlg~new( , , "", "ooRexx Movie Ticket Selectitron")
  intro~setSize(outPageSize)

  p1 = .MoviesDlg~new( , , "", "Movies")
  p1~setSize(inPageSize)

  ...
```

5.3.1.28. translateAccelerator

```
>>--translateAccelerator(--wmMsg--,--keyCode--,--status--,--propSheet--)-------><
```

Notifies a property sheet page that a keyboard message has been received. It provides the page an opportunity to do private keyboard accelerator translation.

Arguments:

The arguments are:

wmMsg

One of the following keywords that indicates the type of keyboard message

KEYUP	DEADCHAR	SYSCHAR
KEYDOWN	SYSKEYUP	SYSDEADCHAR
CHAR	SYSKEYDOWN	

KEYUP

This message is posted to the window with the keyboard focus when a nonsystem key is released. A nonsystem key is a key that is pressed when the ALT key is not pressed, or a keyboard key that is pressed when a window has the keyboard focus. The *keyCode* argument is the virtual-key code of the nonsystem key.

KEYDOWN

This message is posted to the window with the keyboard focus when a nonsystem key is pressed. A nonsystem key is a key that is pressed when the ALT key is not pressed. The *keyCode* argument is the virtual-key code of the nonsystem key.

CHAR

This message is posted to the window with the keyboard focus when a WM_KEYDOWN message is translated by the operating system. The *keyCode* argument is the character code of the key.

DEADCHAR

This message is posted to the window with the keyboard focus when a KEYUP message is translated by the operating system. DEADCHAR specifies a character code generated by a dead key. A dead key is a key that generates a character, such as the umlaut (double-dot), that is combined with another character to form a composite character. For example, the umlaut-O character is generated by typing the dead key for the umlaut character, and then typing the O key. The *keyCode* argument is the character code generated by the dead key.

SYSCHAR

This message is posted to the window with the keyboard focus when a SYSKEYDOWN message is translated by the operating system. The *keyCode* argument specifies the character code of a system character key, that is, a character key that is pressed while the ALT key is down.

SYSDEADCHAR

This message is sent to the window with the keyboard focus when a SYSKEYDOWN message is translated by the operating system. The *keyCode* argument specifies the character code of a system dead key, that is, a dead key that is pressed while holding down the ALT key.

SYSKEYDOWN

This message is posted to the window with the keyboard focus when the user presses the F10 key (which activates the menu bar) or holds down the ALT key and then presses another key. It also occurs when no window currently has the keyboard focus, in this case, the message is sent to the active window. The *keyCode* argument is the virtual-key code of the key being pressed.

SYSKEYUP

This message is posted to the window with the keyboard focus when the user releases a key that was pressed while the ALT key was held down. The *keyCode* argument is the virtual-key code of the key being released.

keyCode

A key code that is dependent on the *wmMsg* argument.

status

A **`Directory`** that further clarifies the status of the keyboard message. The **`Directory`** object will contain the following indexes where the value of each index will be either **`.true`** or **`.false`**:

RELEASED	ISEXTENDED	SHIFTHELD
WASDOWN	ALTHELD	CONTROLHELD

RELEASED

True if the key was released, otherwise false.

WASDOWN

True if the key was down prior to this keyboard message, otherwise false.

ISEXTENDED

True if the key specified by this keyboard message is an extended key, otherwise false.

ALTHELD

True if the Alt key was held down at the time of this message, otherwise false.

SHIFTHELD

True if the Shift key was held down at the time of this message, otherwise false.

CONTROLHELD

True if the Control key was held down at the time of this message, otherwise false.

propSheet

This argument is set to the Rexx property sheet dialog object that owns this property sheet page.

Return value:

The Rexx programmer returns one of the *constants* PSNRET_NOERROR or PSNRET_MESSAGEHANDLED. Return PSNRET_MESSAGEHANDLED to indicate to the OS that no further processing is necessary or return PSNRET_NOERROR to request normal processing from the OS. The default implementation returns PSNRET_NOERROR.

Remarks:

The Microsoft documentation does not specify exactly which keyboard messages are passed on through the TRANSLATEACCELERATOR notification, it simply says *keyboard messages*. The documentation above for the *wmMsg* argument lists possible keyboard messages, but it may not be the case that all of the listed messages are in fact passed on to the event handler.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows an over-ride of the translateAccelerator() event handle that prints out the arguments and some sample output:

```
::method translateAccelerator
  use arg msg, keyCode, d, propSheet
  say msg keyCode
  say 'Released:' d~released 'was down:' d~wasDown 'is extended:' d~isExtended
  say 'Alt held:' d~altHeld 'shift held:' d~shiftHeld 'control held:' d~controlHeld
  say

  return self~PSNRET_NOERROR

/* Output for a shift-M keystroke, i.e. typing capital M. Note the repeated messages
   for the shift key being held down:

KEYDOWN 16
Released: 0 was down: 0 is extended: 0
Alt held: 0 shift held: 1 control held: 0

KEYDOWN 16
Released: 0 was down: 1 is extended: 0
Alt held: 0 shift held: 1 control held: 0

KEYDOWN 16
Released: 0 was down: 1 is extended: 0
Alt held: 0 shift held: 1 control held: 0

KEYDOWN 77
Released: 0 was down: 0 is extended: 0
Alt held: 0 shift held: 1 control held: 0

CHAR 77
Released: 0 was down: 0 is extended: 0
Alt held: 0 shift held: 1 control held: 0

KEYUP 77
Released: 1 was down: 1 is extended: 0
Alt held: 0 shift held: 1 control held: 0

KEYUP 16
Released: 1 was down: 1 is extended: 0
Alt held: 0 shift held: 0 control held: 0

*/
```

5.3.1.29. wizBack

```
>>--wizBack(--propSheet--)-----------------------><
```

Notifies a page that the user has clicked the Back button in a wizard.

Arguments:

The single argument is:

propSheet

This argument is set to the Rexx property sheet dialog object that owns this property sheet page.

Return value:

The Rexx programmer returns 0 to allow the page to be changed to the previous page, -1 to prevent the page from changing, and the page *pageNumber* or page *pageID*, to go to that specific page instead of the previous page. The default implementation returns 0.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example is from some exploratory code. When the user clicks the back button on page four of the wizard, instead of going to page three, the dialog goes back to page one. I.e., the user starts over. There are some say statements to print out information on the screen. The *indexToID* method is used to test that it works. It would have been just as good to simply return the page number (1.)

```
::method wizBack unguarded
  use arg propSheet

  say 'In wizBack page four'
  ret = propSheet~indexToID(1)
  say 'Got return from indexToID() ret:' ret
  return ret

/*  Output might be:

In wizBack page four
Got return from indexToID() ret: 0x00000000004EAF60

*/
```

5.3.1.30. wizFinish

```
>>--wizFinish(--propSheet--)---------------------><
```

Notifies a page that the user has clicked the Finish button in a wizard.

Arguments:

The single argument is:

propSheet

This argument is set to the Rexx property sheet dialog object that owns this property sheet page.

Return value:

The Rexx programmer returns 0 to allow the wizard to finish. To prevent the wizard from finishing, the programmer should return -1 or the resource ID of a dialog control on the page. The default implementation returns 0.

Remarks:

Typically, when the programmer wants to prevent the wizard from finishing it is because of some error in the data the user entered on one of the wizard pages. In this case it is nice to set the focus to the first dialog control on the page that needs correcting. If the *wizFinish* method returns -1, then the programmer has no control over which dialog control will have the focus. It is recommended to always return the resource ID of a dialog control when preventing the wizard from finishing.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows part of a wizFinish() over-ride. The method does some checking and determines that the credit card number entered (or not entered at all) is not valid. A message box will inform the user of the problem and then returns the resource ID of the edit control where the user enters the credit card number. This prevents the wizard from finishing and places the focus on the edit control so that the user can correct the number:

```
::method wizFinish unguarded
  use arg propSheet

  ...

  chkCreditCard = self~newCheckBox(IDC_CK_CREDIT)
  editCCNumber = self~newEdit(IDC_EDIT_CC_NUMBER)

  ...

  if chkCreditCard~checked then do
    ...
    ccNumber = editCCNumber~getText~strip
    if ccNumber~length <> 12 then do
      -- put up message box informing user
      return IDC_EDIT_CC_NUMBER
    end
  end

  ...

  return 0
```

5.3.1.31. wizNext

```
>>--wizNext(--propSheet--)-----------------------><
```

Notifies a page that the user has clicked the Next button in a wizard.

Arguments:

The single argument is:

propSheet

This argument is set to the Rexx property sheet dialog object that owns this property sheet page.

Return value:

The Rexx programmer returns 0 to allow the page to be changed to the next page, -1 to prevent the page from changing, and the page *pageNumber* or page *pageID*, to go to that specific page instead of the next page. The default implementation returns 0.

Details

Raises syntax errors when incorrect usage is detected.

5.3.1.32. validate

```
>>--validate(--isOkButton--,--propSheet--)------><
```

The *validate* method is invoked automatically by the ooDialog framework from the default implementation of the *apply* event handler. It is meant to be over-ridden by the Rexx progammer and provides a place to validate the data entered by the user in a page. It is similar in purpose to the *dialog* object's *validate* method.

Arguments:

The arguments are:

isOkButton

The *validate* method is meant to be invoked from the *apply* event handler. The *isOkButton* argument is the *isOkBtn* argument passed to the *apply* event handler.

The argument is true if the user clicked the OK button. It is also set to true if the *cancelToClose* method had inovked on the property sheet and the user clicked the Close button. It is set to false if the user clicked the Apply button.

propSheet

This argument is set to the Rexx property sheet dialog object that owns this property sheet page.

Return value:

The *validate* method should return one of the 3 *constant*) values: PSNRET_NOERROR, PSNRET_INVALID, or PSNRET_INVALID_NOCHANGEPAGE. See the remarks section.

Remarks:

Recall that the *apply* method handles the notification sent to every page in the property sheet that the user has clicked the OK, Close, or Apply button and wants all changes to take effect. The purpose of that notification, and therefore the *validate* method, is to give the application a chance to check that any data entered by the user is of the right form and prevent the dialog from closing if the user needs to correct any data. The default *apply* implementation returns the value returned from this *validate* method. The default implementation returns PSNRET_NOERROR.

Therefore, a better strategy than over-riding the *apply* method, might be to over-ride the *validate* method. In the over-ride, return PSNRET_INVALID or PSNRET_INVALID_NOCHANGEPAGE to prevent the dialog from closing and return PSNRET_NOERROR to allow the dialog to close. The advantage to this is that the *apply* method also handles some internal house keeping when the return is PSNRET_NOERROR. By over-riding *validate* instead of *apply*, the Rexx programmer has fewer details to take care of.

Example:

This example over-rides the *validate* method for the page and checks that the user has filled in the last name field. If not, PSNRET_INVALID is returned to prevent the dialog from proceeding and to have the focus returned to the page. If the user did fill in the last name and the other checks pass, PSNRET_NOERROR is returned:

```
::method validate private unguarded
  use arg isOkButton, propSheet

  if self~newEdit(IDC_EDIT_LNAME)~getText~strip == "" then do
    msg = "The last name field must be filled in"
    title = "Job Application Error"
    ret = MessageDialog(msg, self~hwnd, title, "OK", "WARNING")
    return self~PSNRET_INVALID
  end

  ...
  return self~PSNRET_NOERROR
```

5.3.1.33. validatePage

```
>>--validatePage(--propSheet--)------------------><
```

The *validatePage* method is invoked automatically by the ooDialog framework from the default implementation of the *killActive* event handler. It is meant to be over-ridden by the Rexx progammer and provides a place to validate the data entered by the user in a page. It is similar in purpose to the *dialog* object's *validate* method.

Arguments:

The single argument is:

propSheet

This argument is set to the Rexx property sheet dialog object that owns this property sheet page.

Return value:

The Rexx programmer should return **`.true`** to signal that the data entered for the page is valid and **`.false`** to signal that the data on the page is not valid. The default implementation of *killActive* directly returns *validatePage*. Therefore, if *killActive* is not over-ridden, the progammer must return **`.true`** to allow the page to lose activation and **`.false`** to prevent the page from losing the activation.

Remarks:

A common strategy might be to over-ride *validatePage* and do all page validation there. When using this strategy, the default implementation of *killActive* would be left alone. The programmer would be careful to return either **`.true`** or **`.false`** from *validatePage*.

On the other hand, if the programmer were to over-ride *killActive*, the implementation of *killActive* would be free to interpret the return from *validatePage* as desired, or not invoke *validatePage* at all.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example over-rides the implementation of *validatePage* and returns **`.true`** or **`.false`** to indicate if the page is valid or invalid. The default implementation of *killActive* is left alone:

```
::method validatePage private unguarded
  expose editLName
  use arg propSheet

  if editLName~getText~strip~length == 0 then do
    msg   = "The last name field must be filled in."
    title = "Acme Employment Application"

    ret = MessageDialog(msg, self~hwnd, title, "OK", "WARNING")

    editLName~assignFocus
    return .false
  end
  return .true
```

5.3.2. RcPSPDialog Class

The **`RcPSPDialog`** class provides the basis for a page dialog where the dialog *template* is defined within a resource *script* file. It is a subclass of the *RcDialog*.

5.3.2.1. new (Class method)

```
>>-init(--rc--,--id-+---------+-+---------+-+---------+-+---------+-+---------+-)--><
                    +-,-data.-+ +-,-hFile-+ +-,-pOpts-+ +-,-cOpts-+ +-,-exptd-+
```

Instantiates a new **`RcPSPDialog`** object. A **`RcPSPDialog`** object can be used as a page in a *PropertySheetDialog* sheet dialog.

Arguments:

The arguments when creating a new dialog instance of this class are:

rc [required]

The file name of the resource script containing the dialog template.

id [required]

The resource ID of the dialog template. This may be numeric or *symbolic*.

data. [optional]

A *dialog data stem* variable (don't forget the trailing period) that contains data used to initialize the underlying dialog's controls.

hFile [optional]

The name of a *file*) containing symbolic ID defines for resource IDs.

pOpts [optional]

A list of 0 or more of the following page option keywords separated by spaces, case is not significant:

AEROPAGE	PREMATURE	USEHEADERTITLE
HASHELP	RTLREADING	USEREFPARENT
HIDEHEADER	SMALL	USETITLE
LARGE	USEFUSIONCONTEXT	
MEDIUM	USEHEADERSUBTITLE	

AEROPAGE

Specifies that this page is for use in an Aero wizard. All property sheet pages that are used in Aero wizard dialogs need to specify this keyword to ensure that the page is initialized correctly.

HASHELP

Enables the property sheet Help button when the page is active. This keyword is not supported when using the Aero-style wizard.

HIDEHEADER

Causes the wizard property sheet to hide the header area when the page is selected. If a watermark has been provided, it will be painted on the left side of the page. This flag should be set for welcome and completion pages, and omitted for interior pages. This keyword is not supported when using the Aero-style wizard.

LARGE

Specifies that the initial size of the page dialog is the Microsoft recommended size for a large property sheet page. This sets the *cx* and *cy* attributes of the page to the *constant*) values PROP_LG_CXDLG and PROP_LG_CYDLG respectively.

Note that this keyword is really only relevant in a **`UserPSPDialog`**. In a **`RcPSPDialog`** the *cx* and *cy* attributes are over-written with the width and height specified in the resource script file. In a **`ResPSPDialog`** the size of the dialog template is set when the resource is compiled and can not be changed so the *cx* and *cy* attributes have no meaning.

MEDIUM

Specifies that the initial size of the page dialog is the Microsoft recommended size for a medium property sheet page. This sets the *cx* and *cy* attributes of the page to the *constant*) values PROP_MED_CXDLG and PROP_MED_CYDLG respectively. This is the default if the SMALL or LARGE keywords are not used.

Note that this keyword is really only relevant in a **`UserPSPDialog`**. In a **`RcPSPDialog`** the *cx* and *cy* attributes are over-written with the width and height specified in the resource script file. In a **`ResPSPDialog`** the size of the dialog template is set when the resource is compiled and can not be changed so the *cx* and *cy* attributes have no meaning.

PREMATURE

Causes the page to be created when the property sheet is created. If this flag is not specified, the page will not be created until it is selected the first time. This keywork is not supported when using the Aero-style wizard.

RTLREADING

Reverses the direction in which the page title is displayed. Normal windows display all text, including the page title, left-to-right (LTR). For languages such as Hebrew or Arabic that read right-to-left (RTL), a window can be mirrored and all text will be displayed RTL. If RTLREADING is set, the page title will instead read RTL in a normal parent window, and LTR in a mirrored parent window.

SMALL

Specifies that the initial size of the page dialog is the Microsoft recommended size for a small property sheet page. This sets the *cx* and *cy* attributes of the page to the *constant*) values PROP_SM_CXDLG and PROP_SM_CYDLG respectively.

Note that this keyword is really only relevant in a **`UserPSPDialog`**. In a **`RcPSPDialog`** the *cx* and *cy* attributes are over-written with the width and height specified in the resource script file. In a **`ResPSPDialog`** the size of the dialog template is set when the resource is compiled and can not be changed so the *cx* and *cy* attributes have no meaning.

USEFUSIONCONTEXT

Use an activation context. This keyword is provided for completeness. In the current implementation of ooDialog there is no way for the programmer to provide the activation context. Future versions of ooDialog may be enhanced in a way that would give meaning to this keyword.

USEHEADERSUBTITLE

Displays the string value of the *headerSubtitle* attribute as the subtitle in the header of a Wizard97 interior page. This keyword is ignored when the HIDEHEADER keyword is used.

USEHEADERTITLE

Displays the string value of the *headerTitle* attribute as the title in the header of a Wizard97 interior page. This keyword is ignored when the HIDEHEADER keyword is used.

USEREFPARENT

Maintains a reference count for the lifetime of the property sheet page. This keyword is present for completeness. It will serve no purpose in the current implementation of ooDialog. Future versions of ooDialog may be enhanced in a way where this keyword would have meaning.

USETITLE

Specifies that the *pageTitle* attribute will be used as the title of the page, rather than the title stored in the dialog template for the page. This keyword is not supported when using the Aero-style wizard.

In general, the ooDialog framework will set this flag correctly and the Rexx programmer does not need to worry about it. *Except*, for the *RcPSPDialog* page object. In this case, the title for the page will be the dialog title as specfied in the resource script, even if the *pageTitle* attribute is set by the programmer. For **`RcPSPDialog`** pages, specify the USETITLE keyword to use the *pageTitle* attribute as the title for the page.

cOpts [optional]

Zero or more of the following connect option keywords, separated by blanks:

CONNECTBUTTONS

Each push *Button* in the underlying dialog has the CLICKED *event* notification connected automatically to a method in the Rexx dialog object. This is the same as using the *connectButtonEvent*() method for the CLICKED notification. The name for the method is generated automatically by the ooDialog framework. The method name is the button label with all spaces, ampersands, colons, and trailing ... characters removed.

CONNECTRADIOS

Similar to CONNECTBUTTONS, this option connects the CLICKED event notification from each *RadioButton* button to a method in the Rexx dialog object. Again, this is the same as using the *connectButtonEvent* method. For radio buttons, the generated method name is the button label with all spaces, ampersands, colons, and trailing ... characters removed. The resulting text is then **prepend** with the text **`ID`**.

CONNECTCHECKS

Exactly the same as CONNECTRADIOS, for check *CheckBox* controls. The object method name is generated in the same way as it is for radio buttons. That is, the method name is the button label, with all spaces, ampersands, colons, and trailing ... characters removed. Which is then **prepended** with the text **`ID`**.

exptd [optional]

This is the maximum number of dialog controls expected in the dialog template. It serves the same purpose as the *expected* argument in the *create*() method of the **`UserDialog`**. The default value for this argument is 200.

Example:

This example creates 5 **RcPSPDialog** objects and then uses them as the 5 pages of a *PropertySheetDialog*:

```
  -- Create the 5 dialog pages.
  p1 = .ListViewDlg~new("rc\PropertySheetDemo.rc", IDD_LISTVIEW_DLG)
  p2 = .TreeViewDlg~new("rc\PropertySheetDemo.rc", IDD_TREEVIEW_DLG)
  p3 = .ProgressBarDlg~new("rc\PropertySheetDemo.rc", IDD_PROGRESSBAR_DLG)
  p4 = .TrackBarDlg~new("rc\PropertySheetDemo.rc", IDD_TRACKBAR_DLG)
  p5 = .TabDlg~new("rc\PropertySheetDemo.rc", IDD_TAB_DLG)

  pages = .array~of(p1, p2, p3, p4, p5)
  propDlg = .PropertySheetDemoDlg~new(pages, "NOAPPLYNOW", "ooRexx Property Sheet with
 Controls")

  ...

::requires "ooDialog.cls"
```

5.3.3. ResPSPDialog Class

The **ResPSPDialog** class provides the basis for a page dialog where the dialog *template* is loaded from a binary resource file. It is a subclass of the *ResDialog*.

5.3.3.1. new (Class method)

```
>>--new(--module--,--id--+------------+--+----------+--+-----------+--)------><
                         +-,--dlgData.-+  +-,--hFile-+  +-,-pageOpts-+
```

Instantiates a new **ResPSPDialog** object. A **ResPSPDialog** can be used as a page in a *PropertySheetDialog*.

Arguments:

The arguments when creating a new dialog instance of this class are:

module [required]

The file name of the executable module (a DLL or EXE) in which the resource (the compiled dialog template) is located.

id [required]

The resource ID of the dialog template. This may be numeric or *symbolic*. The resource ID is assigned to the dialog template when it was compiled.

dlgData. [optional]

A *dialog data stem* variable (don't forget the trailing period) that contains data used to initialize the underlying dialog's controls.

hFile [optional]

The name of a *file*) containing symbolic ID defines for resource IDs.

pageOpts [optional]

A list of 0 or more of the following page option keywords separated by spaces, case is not significant:

AEROPAGE	PREMATURE	USEHEADERTITLE

HASHELP	RTLREADING	USEREFPARENT
HIDEHEADER	SMALL	USETITLE
LARGE	USEFUSIONCONTEXT	
MEDIUM	USEHEADERSUBTITLE	

AEROPAGE

Specifies that this page is for use in an Aero wizard. All property sheet pages that are used in Aero wizard dialogs need to specify this keyword to ensure that the page is initialized correctly.

HASHELP

Enables the property sheet Help button when the page is active. This keyword is not supported when using the Aero-style wizard.

HIDEHEADER

Causes the wizard property sheet to hide the header area when the page is selected. If a watermark has been provided, it will be painted on the left side of the page. This flag should be set for welcome and completion pages, and omitted for interior pages. This keyword is not supported when using the Aero-style wizard.

LARGE

Specifies that the initial size of the page dialog is the Microsoft recommended size for a large property sheet page. This sets the *cx* and *cy* attributes of the page to the *constant*) values PROP_LG_CXDLG and PROP_LG_CYDLG respectively.

Note that this keyword is really only relevant in a **`UserPSPDialog`**. In a **`RcPSPDialog`** the *cx* and *cy* attributes are over-written with the width and height specified in the resource script file. In a **`ResPSPDialog`** the size of the dialog template is set when the resource is compiled and can not be changed so the *cx* and *cy* attributes have no meaning.

MEDIUM

Specifies that the initial size of the page dialog is the Microsoft recommended size for a medium property sheet page. This sets the *cx* and *cy* attributes of the page to the *constant*) values PROP_MED_CXDLG and PROP_MED_CYDLG respectively. This is the default if the SMALL or LARGE keywords are not used.

Note that this keyword is really only relevant in a **`UserPSPDialog`**. In a **`RcPSPDialog`** the *cx* and *cy* attributes are over-written with the width and height specified in the resource script file. In a **`ResPSPDialog`** the size of the dialog template is set when the resource is compiled and can not be changed so the *cx* and *cy* attributes have no meaning.

PREMATURE

Causes the page to be created when the property sheet is created. If this flag is not specified, the page will not be created until it is selected the first time. This keywork is not supported when using the Aero-style wizard.

RTLREADING

Reverses the direction in which the page title is displayed. Normal windows display all text, including the page title, left-to-right (LTR). For languages such as Hebrew or Arabic that read right-to-left (RTL), a window can be mirrored and all text will be displayed RTL. If RTLREADING is set, the page title will instead read RTL in a normal parent window, and LTR in a mirrored parent window.

SMALL

Specifies that the initial size of the page dialog is the Microsoft recommended size for a small property sheet page. This sets the *cx* and *cy* attributes of the page to the *constant*) values PROP_SM_CXDLG and PROP_SM_CYDLG respectively.

Note that this keyword is really only relevant in a **`UserPSPDialog`**. In a **`RcPSPDialog`** the *cx* and *cy* attributes are over-written with the width and height specified in the resource script file. In a **`ResPSPDialog`** the size of the dialog template is set when the resource is compiled and can not be changed so the *cx* and *cy* attributes have no meaning.

USEFUSIONCONTEXT

Use an activation context. This keyword is provided for completeness. In the current implementation of ooDialog there is no way for the programmer to provide the activation context. Future versions of ooDialog may be enhanced in a way that would give meaning to this keyword.

USEHEADERSUBTITLE

Displays the string value of the *headerSubtitle* attribute as the subtitle in the header of a Wizard97 interior page. This keyword is ignored when the HIDEHEADER keyword is used.

USEHEADERTITLE

Displays the string value of the *headerTitle* attribute as the title in the header of a Wizard97 interior page. This keyword is ignored when the HIDEHEADER keyword is used.

USEREFPARENT

Maintains a reference count for the lifetime of the property sheet page. This keyword is present for completeness. It will serve no purpose in the current implementation of ooDialog. Future versions of ooDialog may be enhanced in a way where this keyword would have meaning.

USETITLE

Specifies that the *pageTitle* attribute will be used as the title of the page, rather than the title stored in the dialog template for the page. This keyword is not supported when using the Aero-style wizard.

In general, the ooDialog framework will set this flag correctly and the Rexx programmer does not need to worry about it. *Except*, for the *RcPSPDialog* page object. In this case, the title for the page will be the dialog title as specfied in the resource script, even if the *pageTitle* attribute is set by the programmer. For **`RcPSPDialog`** pages, specify the USETITLE keyword to use the *pageTitle* attribute as the title for the page.

Example:

This sample code snippet shows the instantiation of 5 **`ResPSPDialog`** objects. These dialogs are then suitable to be used as the pages of a **`PropertySheetDialog`** object:

```
  t1 = .ListViewDlg~new("rc\PropertySheetDemo.dll", IDD_LISTVIEW_DLG)
  t2 = .TreeViewDlg~new("rc\PropertySheetDemo.dll", IDD_TREEVIEW_DLG)
  t3 = .ProgressBarDlg~new("rc\PropertySheetDemo.rc", IDD_PROGRESSBAR_DLG)
  t4 = .TrackBarDlg~new("rc\PropertySheetDemo.dll", IDD_TRACKBAR_DLG)
  t5 = .TabDlg~new("rc\PropertySheetDemo.dll", IDD_TAB_DLG)

  pages = .array~of(t1, t2, t3, t4, t5)

...
```

```
::requires "ooDialog.cls"
```

5.3.4. UserPSPDialog Class

The **`UserPSPDialog`** class provides the basis for a page dialog where the dialog *template* is built dynamically in the program using the *create...* methods of the *UserDialog*. It is a subclass of the **`UserDialog`**.

5.3.4.1. new (Class method)

```
>>--new(--+----------+--+--------------+--+-----------+--+---------+--------->
          +-dlgData.-+  +-,--headerFile-+  +-,-pageOpts-+  +-,-title-+

>----+-----------+--+-----------+--+-----------+--)------------------------><
     +-,-fontName-+  +-,-fontSize-+  +-,-expected-+
```

Instantiates a new **`UserPSPDialog`** object. This dialog can then be used as a page in a *PropertySheetDialog*.

Arguments:

The arguments when creating a new dialog instance of this class are:

dlgData. [optional]

A *dialog data stem* variable (don't forget the trailing period) that contains data used to initialize the underlying dialog's controls.

hFile [optional]

The name of a *file*) containing symbolic ID defines for resource IDs.

pageOpts [optional]

A list of 0 or more of the following page option keywords separated by spaces, case is not significant:

AEROPAGE	PREMATURE	USEHEADERTITLE
HASHELP	RTLREADING	USEREFPARENT
HIDEHEADER	SMALL	USETITLE
LARGE	USEFUSIONCONTEXT	
MEDIUM	USEHEADERSUBTITLE	

AEROPAGE

Specifies that this page is for use in an Aero wizard. All property sheet pages that are used in Aero wizard dialogs need to specify this keyword to ensure that the page is initialized correctly.

HASHELP

Enables the property sheet Help button when the page is active. This keyword is not supported when using the Aero-style wizard.

HIDEHEADER

Causes the wizard property sheet to hide the header area when the page is selected. If a watermark has been provided, it will be painted on the left side of the page. This flag should be set for welcome and completion pages, and omitted for interior pages. This keyword is not supported when using the Aero-style wizard.

LARGE

Specifies that the initial size of the page dialog is the Microsoft recommended size for a large property sheet page. This sets the *cx* and *cy* attributes of the page to the *constant*) values PROP_LG_CXDLG and PROP_LG_CYDLG respectively.

Note that this keyword is really only relevant in a **`UserPSPDialog`**. In a **`RcPSPDialog`** the *cx* and *cy* attributes are over-written with the width and height specified in the resource script file. In a **`ResPSPDialog`** the size of the dialog template is set when the resource is compiled and can not be changed so the *cx* and *cy* attributes have no meaning.

MEDIUM

Specifies that the initial size of the page dialog is the Microsoft recommended size for a medium property sheet page. This sets the *cx* and *cy* attributes of the page to the *constant*) values PROP_MED_CXDLG and PROP_MED_CYDLG respectively. This is the default if the SMALL or LARGE keywords are not used.

Note that this keyword is really only relevant in a **`UserPSPDialog`**. In a **`RcPSPDialog`** the *cx* and *cy* attributes are over-written with the width and height specified in the resource script file. In a **`ResPSPDialog`** the size of the dialog template is set when the resource is compiled and can not be changed so the *cx* and *cy* attributes have no meaning.

SMALL

Specifies that the initial size of the page dialog is the Microsoft recommended size for a small property sheet page. This sets the *cx* and *cy* attributes of the page to the *constant*) values PROP_SM_CXDLG and PROP_SM_CYDLG respectively.

Note that this keyword is really only relevant in a **`UserPSPDialog`**. In a **`RcPSPDialog`** the *cx* and *cy* attributes are over-written with the width and height specified in the resource script file. In a **`ResPSPDialog`** the size of the dialog template is set when the resource is compiled and can not be changed so the *cx* and *cy* attributes have no meaning.

PREMATURE

Causes the page to be created when the property sheet is created. If this flag is not specified, the page will not be created until it is selected the first time. This keywork is not supported when using the Aero-style wizard.

RTLREADING

Reverses the direction in which the page title is displayed. Normal windows display all text, including the page title, left-to-right (LTR). For languages such as Hebrew or Arabic that read right-to-left (RTL), a window can be mirrored and all text will be displayed RTL. If RTLREADING is set, the page title will instead read RTL in a normal parent window, and LTR in a mirrored parent window.

USEFUSIONCONTEXT

Use an activation context. This keyword is provided for completeness. In the current implementation of ooDialog there is no way for the programmer to provide the activation context. Future versions of ooDialog may be enhanced in a way that would give meaning to this keyword.

USEHEADERSUBTITLE

Displays the string value of the *headerSubtitle* attribute as the subtitle in the header of a Wizard97 interior page. This keyword is ignored when the HIDEHEADER keyword is used.

USEHEADERTITLE

Displays the string value of the *headerTitle* attribute as the title in the header of a Wizard97 interior page. This keyword is ignored when the HIDEHEADER keyword is used.

USEREFPARENT

Maintains a reference count for the lifetime of the property sheet page. This keyword is present for completeness. It will serve no purpose in the current implementation of ooDialog. Future versions of ooDialog may be enhanced in a way where this keyword would have meaning.

USETITLE

Specifies that the *pageTitle* attribute will be used as the title of the page, rather than the title stored in the dialog template for the page. This keyword is not supported when using the Aero-style wizard.

In general, the ooDialog framework will set this flag correctly and the Rexx programmer does not need to worry about it. *Except*, for the *RcPSPDialog* page object. In this case, the title for the page will be the dialog title as specfied in the resource script, even if the *pageTitle* attribute is set by the programmer. For **`RcPSPDialog`** pages, specify the USETITLE keyword to use the *pageTitle* attribute as the title for the page.

title [optional]

The text to use for this page's title. The page title is used as the text for the tab label for this page in the property sheet *PropertySheetDialog*. This argument sets the *pageTitle* attribute of this property sheet page and automatically adds the USETITLE keyword to the *pageOpts* argument. If this arugment is omitted and the *pageTitle* attribute is not set by the programmer, then a label is constructed automatically by the ooDialog framework. The label will consist of the word *Page* followed by the page number for this page. For example, *Page 2* or *Page 5*, dependent on the actual page number for this page.

fontName [optional]

The name of the font to be used by the dialog for all text. If this argument is omitted, the default dialog font is used. The programmer can change this default by using *setDefaultFont* class method. If the default font has not been changed, ooDialog uses a symbolic font name provided by Microsoft. This symbolic name signals the operating system to use the appropriate dialog font for the specific version of Windows in use. I.e., the dialog font under Windows 2000 might be different then the font under Windows XP.

fontSize [optional]

The point size of the font used by the dialog. When this argument is omitted, the default dialog font size is used, in the same manner as the font name.

expected [optional]

This argument specifies the maximum number of dialog controls the dialog template is expected to contain. Since the dialog template is constructed in memory, ooDialog needs to have some idea of the size of the system resources to reserve for the template. The default value is 200.

This value is just a means of doing a rough estimate of the memory needed for the template. The amount of memory for each dialog control varies depending on how much, if any, text is associated with the control. If your dialog has more than 200 controls, you should probably set the expected value larger.

As the dialog template is being constructed, the ooDialog framework will detect if the template is about to extend past its allocated storage. If so, an error condition is raised. The condition text will look similar to:

```
Error 98 running C:\Program Files\ooDialog.cls line xxxx:  Execution error
Error 98.900:  the storage allocated for the dialog template is too small
```

If this condition is raised, the programmer needs to use a larger value for the *expected* argument.

Resizable Dialogs

In the early versions of Windows, dialogs in general were always fixed in size. Even today, most dialogs are fixed in size. However, the use of resizable dialogs has become very common. Giving the user the ability to resize the dialogs in your program is a nice feature. Especially if the dialogs contain list boxes, list-views, or tree-views.

ooDialog provides two basic ways to make dialogs resizable. One is through the *DlgAreaU* class and the other is through the *ResizingAdmin* class.

6.1. DlgArea Class

The **`DlgArea`** class provides assistance in laying out the dialog controls in a dynamically defined dialog. The class defines an area within the *client area* of a *UserDialog*. It is a helper class for the *defineDialog* method of a **`UserDialog`** and has no effect on any *underlying* window.

Note that the **`DlgArea`** class does not, in and of itself, produce a resizable dialog. It is the use of its subclass, the *DlgAreaU* class that produces a resizable dialog.

The methods and attributes of the **`DlgArea`** provide the coordinates of, and within, the defined area. All coordinates are specified as *client area* coordinates. That is, all coordinates are relative to the origin (0, 0) of the client area of the dialog.

The following figure shows the measurement attributes used to position a **`DlgArea`** on the client area of a **`UserDialog`**.

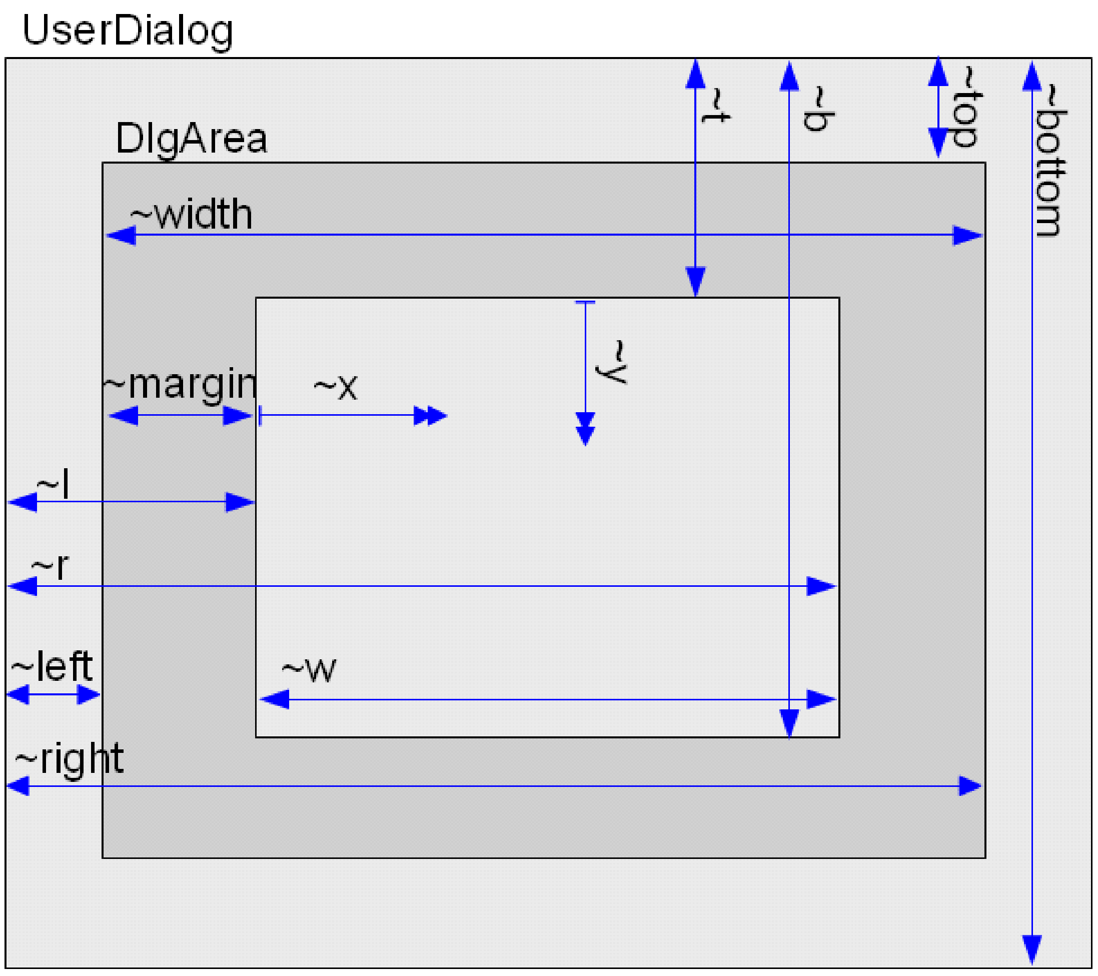


Figure 6.1. DlgArea Measurements

6.1.1. Method Table

The following table lists the class methods, attributes, and instance methods of the **`DlgArea`** class:

Table 6.1. DlgArea Class Method Reference

Method	Description
Class Methods	**Class Methods**
init	Instantiates a new **`DlgArea`** object.
Attributes	**Attributes**
b	Reflects the y coordinate, in dialog units, of the bottom margin of the dialog area.
height	Reflects the height of the dialog area in dialog units.
l	Reflects the x coordinate, in dialog units, of the left margin of the area.
left	Reflects the x coordinate, in dialog units, of the left edge of the area.
margin	Reflects the size, in dialog units, of the dialog area margin.
r	Reflects the x coordinate, in dialog units, of the right margin of the area.
t	Reflects the y coordinate, in dialog units, of the top margin of the area.
top	Reflects the y coordinate, in dialog units, of the top edge of the area.
width	Reflects the width of the dialog area in dialog units.

Method	Description
Instance Methods	**Instance Methods**
cx	Returns a width, in dialog units, relative to the width of the dialog area.
cy	Returns a height, in dialog units, relative to the height of the dialog area.
h	Returns a height, in dialog units, relative to the height of the dialog area.
hr	Returns the remaining height, in dialog units, between the last y invocation and the bottom margin of the area.
right	Returns the x coordinate, in dialog units, of the right edge of the area.
w	Returns a width, in dialog units, relative to the width of the dialog area.
wr	Returns the remaining width, in dialog units, between the last x invocation and the right margin of the area.
x	Returns a x coordinate relative to the client area of the dialog by specifying an offset relative to the defined dialog area.
y	Returns a y coordinate relative to the client area of the dialog by specifying an offset relative to the defined dialog area.

6.1.2. init

```
>>--new(--x--,--y--,--width--,--height--+-------------+--)------><
                                        +--,--margin--+
```

Instantiates a new **DlgArea** object.

Arguments:

The arguments are:

x [required]

The x coordinate for the area to be defined. The coordinate is specified in *dialog unit*s and is relative to the top left corner of the *client area* of the **UserDialog**.

y [required]

The y coordinate for the area to be defined. The coordinate is specified in dialog units and is relative to the top left corner of the client area of the **UserDialog**.

width [required]

The width, in dialog units, of the area being defined.

height [required]

The height, in dialog units, of the area being defined.

margin [optional]

An inner margin, in dialog units, within the dialog area to be left blank. The default is 5

Example:

This examples defines two dialog areas in a dialog. The first area, *u*, is defined for the entire client area of the dialog. The second area, **b** is defined as a sub-area of the **u** area.

```
u = .dlgArea~new(0, 0, self~sizeX,self~sizeY)
b = .dlgArea~new(u~x("70%"), u~y , u~w("R"), u~h("R"))
```

6.1.3. t (Attribute)

```
>>--t--------------------------------------------><
```

Reflects the y coordinate, in dialog units, of the top margin of the area.

6.1.4. l (Attribute)

```
>>--l--------------------------------------------><
```

Reflects the x coordinate, in dialog units, of the left margin of the area.

6.1.5. b (Attribute)

```
>>--b--------------------------------------------><
```

Reflects the y coordinate, in dialog units, of the bottom margin of the area.

6.1.6. r (Attribute)

```
>>--r--------------------------------------------><
```

Reflects the x coordinate, in dialog units, of the right margin of the area.

6.1.7. top (Attribute)

```
>>--top------------------------------------------><
```

Reflects the y coordinate, in dialog units, of the top edge of the area.

6.1.8. left (Attribute)

```
>>--left-----------------------------------------><
```

Reflects the x coordinate, in dialog units, of the left edge of the area.

6.1.9. width (Attribute)

```
>>--width----------------------------------------><
```

Reflects the width of the dialog area in dialog units.

6.1.10. height (Attribute)

```
>>--height---------------------------------------><
```

Reflects the height of the dialog area in dialog units.

6.1.11. margin (Attribute)

```
>>--margin---------------------------------------><
```

Reflects the size, in dialog units, of the dialog area margin.

6.1.12. cx

```
>>--cx(--+---------+--)--------------------------><
         +--flag---+
```

Returns a width, in dialog units, relative to the width of the dialog area. The *cx* and *w* methods are synonomous.

Arguments:

The single argument is:

flag [optional]

The *flag* argument signals what part of the width to return.

If the flag is omitted, then the entire width of the dialog area, excluding the margins, is returned. If flag is not omitted, it should be one of the following symbols:

n%

Where **n** is a number and the percent sign signals that a percentage of the dialog area width should be returned. E.g., **`10%`** would mean to return 10 percent of the dialog area width. **`8%`** would ask for 8 percent of the width, etc..

R

A capital **R** indicates the *remainder* of the dialog area width should be returned. I.e., that portion of the width between the last invocation of the *x* method and the right margin.

Returns:

The return is part, or all, of the dialog area width as specified by the *flag* argument.

Remarks:

Rigours error checking of the argument is not done. If the argument is not recognized it is just ignored and treated as if the argument was omitted.

6.1.13. cy

```
>>--cy(--+---------+--)--------------------------><
         +--flag---+
```

Returns a height, in dialog units, relative to the height of the dialog area. The *cy* and *h* methods are synonomous.

Arguments:

The single argument is:

flag [optional]

The *flag* argument signals what part of the height to return.

If the flag is omitted, then the entire height of the dialog area, excluding the margins, is returned. If flag is not omitted, it should be one of the following symbols:

n%

Where **n** is a number and the percent sign signals that a percentage of the dialog area height should be returned. E.g., **4%** would mean to return 4 percent of the dialog area height **20%** would ask for 20 percent of the height, etc..

R

A capital **R** indicates the *remainder* of the dialog area height should be returned. I.e., that portion of the height between the last invocation of the *y* method and the bottom margin.

Returns:

The return is part, or all, of the dialog area height as specified by the *flag* argument.

Remarks:

Rigours error checking of the argument is not done. If the argument is not recognized it is just ignored and treated as if the argument was omitted.

6.1.14. h

Returns a height, in dialog units, relative to the height of the dialog area. The *h* and *cy* methods are synonomous.

Arguments:

The single argument is:

flag [optional]

The *flag* argument signals what part of the height to return.

If the flag is omitted, then the entire height of the dialog area, excluding the margins, is returned. If flag is not omitted, it should be one of the following symbols:

n%

Where **n** is a number and the percent sign signals that a percentage of the dialog area height should be returned. E.g., **4%** would mean to return 5 percent of the dialog area height **20%** would ask for 20 percent of the height, etc..

R

A capital **R** indicates the *remainder* of the dialog area height should be returned. I.e., that portion of the height between the last invocation of the *y* method and the bottom margin.

Returns:

The return is part, or all, of the dialog area height as specified by the *flag* argument.

Remarks:

Rigours error checking of the argument is not done. If the argument is not recognized it is just ignored and treated as if the argument was omitted.

6.1.15. hr

```
>>--hr-------------------------------------------><
```

Returns the remaining height, in dialog units, between the last *y* invocation and the bottom margin of the area. This is equivalent to *h*("R").

6.1.16. right

```
>>--right----------------------------------------><
```

Returns the x coordinate, in dialog units, of the right edge of the area.

6.1.17. w

```
>>--w(--+---------+--)---------------------------><
        +--flag---+
```

Returns a width, in dialog units, relative to the width of the dialog area. The *cx* and *w* methods are synonomous.

Arguments:

The single argument is:

flag [optional]

The *flag* argument signals what part of the width to return.

If the flag is omitted, then the entire width of the dialog area, excluding the margins, is returned. If flag is not omitted, it should be one of the following symbols:

n%

Where **`n`** is a number and the percent sign signals that a percentage of the dialog area width should be returned. E.g., **`10%`** would mean to return 10 percent of the dialog area width. **`8%`** would ask for 8 percent of the width, etc..

R

A capital **`R`** indicates the *remainder* of the dialog area width should be returned. I.e., that portion of the width between the last invocation of the *x* method and the right margin.

Returns:

The return is part, or all, of the dialog area width as specified by the *flag* argument.

Remarks:

Rigours error checking of the argument is not done. If the argument is not recognized it is just ignored and treated as if the argument was omitted.

6.1.18. wr

```
>>--wr-------------------------------------------><
```

Returns the remaining width, in dialog units, between the last *x* invocation and the right margin of the area. This is equivalent to *w*("R").

6.1.19. x

```
>>--x(--+----------+--)--------------------------><
        +--offSet--+
```

Returns a x coordinate relative to the *client area* of the dialog by specifying an offset relative to the defined dialog area.

Arguments:

The single argument is:

offset [optional]

If *offset* is omitted, then the x coordinate of the left margin is returned.

Otherwise, *offset* is specified as either a concrete number or as a percentage. To specify a percentage add the percent sign to the number, e.g. for twenty percent use **`20%`**.

In addition, *offset* can be negative or positive. When positive, the offset is an amount to the right of the left margin. When negative, the offset is an amount to the left of the right margin. Both concrete and percentage offsets can be negative.

Returns:

The appropriate *client area* x coordinate, in dialog units, as specified by the *offset* argument.

6.1.20. y

```
>>--y(--+----------+--)--------------------------><
        +--offSet--+
```

Returns a y coordinate relative to the *client area* of the dialog by specifying an offset relative to the defined dialog area.

Arguments:

The single argument is:

offset [optional]

If *offset* is omitted, then the y coordinate of the top margin is returned.

Otherwise, *offset* is specified as either a concrete number or as a percentage. To specify a percentage add the percent sign to the number, e.g. for twenty percent use **`20%`**.

In addition, *offset* can be negative or positive. When positive, the offset is an amount below the top margin. When negative, the offset is an amount above the bottom margin. Both concrete and percentage offsets can be negative.

Returns:

The appropriate *client area* y coordinate, in dialog units, as specified by the *offset* argument.

6.1.21. DlgArea Example

With a **`UserDialog`** we want the top left to be an edit control, with an area for buttons on the right and a status area below.

We decide to give 70% of the width and 90% of the height to the edit control. We leave the margin as the default.

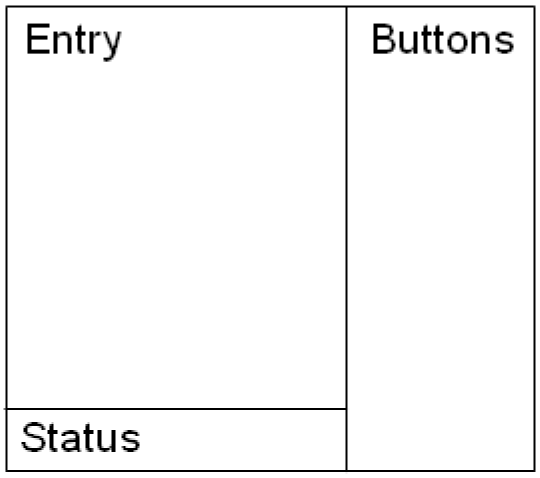


Figure 6.2. DlgArea Plan

Here is what our defineDialog method might look like:

```
/* ------------------------------------------------------------------------- */
::method defineDialog
/* ------------------------------------------------------------------------- */
/* define DlgArea named u as whole of user dialog                            */
u=.dlgArea~new(  0       ,         0,self~SizeX,Self~SizeY)   /* whole dlg   */

/* define DlgArea named e within DlgArea u for edit control                  */
e=.dlgArea~new(u~x       ,u~y       ,u~w("70%"),u~h("90%"))

/* define DlgArea named s within DlgArea u for Status Area in remaining height*/
s=.dlgArea~new(u~x       ,u~y("90%"),u~w("70%"),u~hr      )

/* define DlgArea named b within DlgArea u for Button area                   */
b=.dlgArea~new(u~x("70%"),u~y       ,u~wr      ,u~hr      )

/* Edit control coterminous with area e margins                              */
self~createEdit(12, e~x, e~y, e~w, e~h, "multiline", "text")

/* Status Area Coterminous with area s margins                               */
self~createStaticText(11, s~x, s~y, s~w, s~h, , "Status info appears here")

/* Seven buttons evenly spaced at 10% intervals, 9% high                     */
do i = 0 to 6
   self~createPushButton(12+i,b~x,b~y((i * 10)||"%"),b~w,b~h("9%"), ,"Button" i,"Button"||i)
end /* DO */

/* ok button 15 dialog units high at bottom of area b                        */
self~createPushButton(IDOK,b~x,b~y(-15),b~w,15,"Default","OK","OK")
```

Here is the resultant dialog.

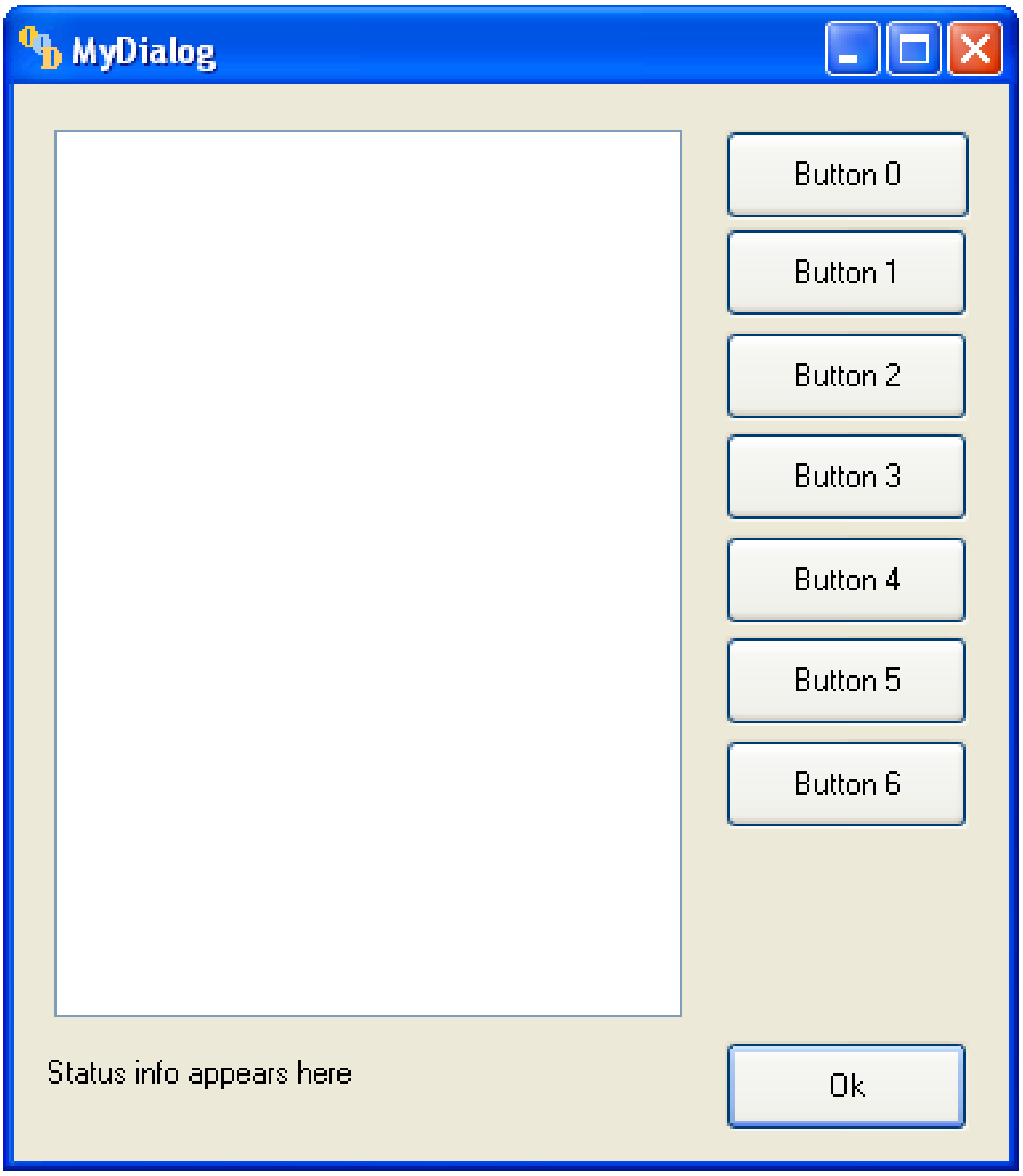


Figure 6.3. Sample DlgArea

6.2. DlgAreaU Class

The **`DlgAreaU`** class is a subclass of the *DlgArea* class that helps with the creation of dynamically resizable dialogs. This class provides assistance in resizing and / or positioning controls when a dialog is resized. When the **`DlgArea`** and **`DlgAreaU`** classes are used together, they provide a convenient way to create resizable dialogs.

Note: The **`DlgAreaU`** class uses the *.Object* class's *source* class method to parse the source code of the dialog's *defineDialog* method. This source is not available if the *rexxc* program is used to tokenize the source code. Therefore, *DlgAreaU* class will **not** work if *rexxc* is used to tokenize the source code.

The other implication of using the *source* method to parse the dialog's *defineDialog* method is that the **`DlgAreaU`** class can only be used with a *UserDialog*. It will not work with any other dialog class, such as the *ResDialog* or *RcDialog* classes.

6.2.1. Method Table

Instances of the **DlgAreaU** class implement the methods listed in the following table.

Table 6.2. DlgAreaU Method Reference

Method	Description
Class Methods	**Class Methods**
new	Instantiates a new **DlgAreaU** object.
Attributes	
correctionFactor	A value used to fix problems in the conversion of pixels to dialog units by adjusting the ratio of original size to change in size.
lastError	Holds the details if an error was encountered when parsing the dialog's defineDialog method.
noMove	Holds a set of the resource IDs of dialog controls not to be moved during a resize event.
noResize	Holds a set of the resource IDs of dialog controls not to be resized during a resize event.
updateOnResize	Controls whether the DlgAreaU object forces the dialog controls to redraw after every resize event.
Instance Methods	**Instance Methods**
noMovePut	Adds a dialog control to the set of dialog controls that will not be moved during the resize event.
noResizePut	Adds a dialog control to the set of dialog controls that will not be resized during the resize event.
resize	Causes the DlgAreaU object to resize and reposition all the dialog controls in the dialog.

6.2.2. new (Class method)

```
>>--new(-dlg--+----------+--+------------+--+--------+-)--------><
              +-,-margin-+  +-,-noResize-+  +-noMove-+
```

The **DlgAreaU** object creates a dialog area coterminous with the calling dialog. It is a subclass of the *DlgArea* class and therefore inherits all the methods and attributes of that class.

Arguments:

The arguments are:

dlg
: The resizable dialog you are creating a dialog area for. This dialog must be subclassed from a *UserDialog*.

margin
: The **DlgArea** *margin*.

noResize
: A **.Set** object containing the resource *resource ID*s of the dialog controls that should **not** be resized during the resize event. The IDs may be numeric or *symbolic*.

noMove
: A **.Set** object containing the resource *resource IDs* of the dialog controls that should **not** be moved during the resize event. The IDs may be numeric or *symbolic*.

6.2.3. correctionFactor (Attribute)

```
>>--correctionFactor-----------------------------><

>>--correctionFactor=num-------------------------><
```

An attribute containing the value used to adjust the ratio of the top & left margins to the bottom & right margins. (The default correction factor is set to 1.05.) The Possible *Problems* section has additional details on using the *correctionFactor* attribute.

6.2.4. lastError (Attribute)

```
>>--lastError------------------------------------><

>>--lastError = errMsg---------------------------><
```

An attribute holding the details if an error was encountered when parsing the calling dialog's *defineDialog* method. The value of the attribute is .nil if no error occurred.

6.2.5. noMove (Attribute)

```
>>--noMove---------------------------------------><

>>--noMove = ids---------------------------------><
```

A **.Set** object holding the resource IDs of the dialog controls that the programmer does not wish to be moved during a resize event. The IDs in the set must be numeric.

The programmer can use *symbolic* IDs for the dialog controls by adding to the set through the *noMovePut* method. Or by passing in a **.Set** object when instantiating a *new* **DlgAreaU** object. If the programmer accesses the set of IDs directly through the *noMove* attribute to add a resource ID, only numeric IDs should be used.

6.2.6. noResize (Attribute)

```
>>--noResize-------------------------------------><

>>--noResize = ids-------------------------------><
```

A **.Set** object holding the resource IDs of the dialog controls that the programmer does not wish to be resized during a resize event. The IDs in the set must be numeric.

The programmer can use *symbolic* IDs for the dialog controls by adding to the set through the *noResizePut* method. Or by passing in a **.Set** object when instantiating a *new* **DlgAreaU** object. If the programmer accesses the set of IDs directly through the *noResize* attribute to add a resource ID, only numeric IDs should be used.

6.2.7. updateOnResize (Attribute)

```
>>--updateOnResize-------------------------------><
>>--updateOnResize = boolean---------------------><
```

An attribute that controls whether the **`DlgAreaU`** object forces the dialog controls to redraw during every resize event. The default value of this attribute is **`.true`**, the dialog controls are forced to redraw on every resize event. Setting this attribute to false will prevent the dialog controls from being forced to redraw.

When the *updateOnResize* attribute is **`.false`**, it becomes the programmer's responsibility to force the dialog controls to redraw when the user has stopped resizing the dialog. This can be down by using the *connectSizeMoveEnded* method. Having the dialog controls redraw only once eliminates flicker while the user is sizing the dialog. However, the dialog controls do not change size or position while the using is sizing the dialog. In the **`samples\oodialog`** directory of the ooRexx installation are two example programs: **`dlgAreaUDemo.rex`** redraws the dialog controls on every resize event. **`dlgAreaUDemoTwo.rex`** only redraws the controls once, when the user has stopped resizing the controls. Contrast the two programs to determine which method best suits the needs or your program.

6.2.8. noMovePut

```
>>--noMovePut(--resourceID--)--------------------><
```

Adds a dialog control to the *noMove* of dialog controls that will not be moved during the resize event.

Arguments:

The single argument is:

resourceID [required]

The resource ID of the dialog control that is not to be moved during the resize event. May be symbolic or numeric.

Return value:

This method does not return a value.

Remarks:

Using this method allows the programmer to use *symbolic* IDs for the dialog control. If the resource ID is put directly into the set through the *noMove* attribute, symbolic IDs will not be recognized.

Details

If a symbolic ID is used and it can not be resolved a syntax error is raised.

6.2.9. noResizePut

```
>>--noResizePut(--resourceID--)------------------><
```

Adds a dialog control to the *noResize* of dialog controls that will not be resized during the resize event.

Arguments:

The single argument is:

resourceID [required]

The resource ID of the dialog control that is not to be resized during the resize event. May be symbolic or numeric.

Return value:

This method does not return a value.

Remarks:

Using this method allows the programmer to use *symbolic* IDs for the dialog control. If the resource ID is put directly into the set through the *noResize* attribute, symbolic IDs will not be recognized.

Details

If a symbolic ID is used and it can not be resolved a syntax error is raised.

6.2.10. resize

```
>>--resize(--dlgObj--,--sizeInfo--)--------------><
```

The *resize* method causes the **`DlgAreaU`** object to resize and reposition all the dialog controls in the *dlgObj* dialog in relation to the *sizeInfo* argument. Normally the dialog is then told to update itself, which redraws the controls. However, the updating is dependent on the value of the *updateOnResize* attribute.

Arguments:

The arguments are:

dlgObj [required]

The dialog which is being resized. The dialog must be the same dialog used to instantiate the *new* **`DlgAreaU`** object.

sizeInfo [required]

The new size of the dialog. Normally, *sizeInfo* comes from the value passed to your resize *event* handler.

Return value:

This method does not return a value.

Remarks:

The most common way to use the **`DlgAreaU`** class is to connect the *connectResize* event to an event handling method in a resizable dialog. Then the *sizeInfo* arg to the event handler is simply passed on to the *resize* method.

Example:

This example shows a typical event handling method in a resizable dialog that makes use of the **`DlgAreaU`** class

```
::method defineDialog
  expose u

  self~connectResize('onResize')

  u = .dlgAreaU~new(self)
  ...


::method onResize unguarded
  expose u
  use arg sizeEvent, sizeInfo

  u~resize(self, sizeInfo)
  return 0
```

6.2.11. Creating resizable Dialogs

You can use the **`DlgAreaU`** class to facilitate creating dynamically resizable dialogs. However the following restrictions apply:

The dialog must be a *UserDialog* with all dialog controls created in the dialog template from within the *defineDialog* method. In the *create* or *createCenter* method, the dialog template must be created with the THICKFRAME option.

You must connect the *connectResize* event to a method in your dialog. Add the following code somewhere in your dialog code. The *init*, *defineDialog* or *initDialog* methods are all a suitable place for the line. The *methodName* argument is the name of your event handling method and can be any appropriate method name.

```
self~connectResize(methodName)
```

Your *defineDialog* method must start with these lines:

```
expose u
u = .dlgAreaU~new(self)
```

Your *defineDialog* method must not reference variables within the *create*/> control method parameters. Although you can use references to *DlgArea* attributes, so

```
  self~createPushButton(IDC_PB, 5, 100, 45, 15, , "MyButton", "Pressed")
  self~createPushButton(IDC_PB_1, b~x, b~y("10%"), b~w, b~h("9%"), , "Button" 1, "Button"||1)
```

work fine, whereas

```
/* Seven buttons evenly spaced at 10% intervals, 9% high                     */
do i = 0 to 6
   self~createPushButton(12+i, b~x, b~y((i * 10)||"%"), b~w, b~h("9%"), , "Button" i,
 "Button"||i)
end /* DO */
```

would fail because the *createPushButton* method references the variable **`i`** within its parameters.

To debug your **`DlgAreaU`** object call/insert the following after the instantiation:

```
if u~lastError \= .nil then call errorDialog u~lastError
```

Your event handler code for the RESIZE event must pass on the second argument of the method to the **`DlgAreaU`** object's *resize* method.

```
/* ------------------------------------------------------------------------- */
::method OnResize
/* ------------------------------------------------------------------------- */
```

```
  expose u
  use arg sizeEvent, sizeInfo

  u~resize(self, sizeInfo)
```

The **`DlgAreaU`** object's *resize* method will then automatically resize & position the dialog controls in your dialog that were added to the dialog template with the following methods. The resizing takes place when the dialog frame is dragged, or the minimize, maximize, or restore buttons are pressed:

Table 6.3. DlgAreaU Automatic Resize Controls

createBitmapbutton	createBlackframe	createBlackrect	createCheckbox
createCombobox	createEdit	createDatetimePicker	createEdit
createEtchedHorizontal	createEtchedVertical	createGrayFrame	createGrayRect
createGroupBox	createListBox	createListView	createMonthCalendar
createPasswordEdit	createProgressBar	createPushButton	createRadioButton
createScrollBar	createStatic	createStaticFrame	createStaticImage
createStaticText	createTab	createTrackBar	createTreeView
createUpDown	createWhiteFrame	createWhiteRect	

The following *create* methods can be used to define dialog controls in a *UserDialog*, but cannot be handled by the **`DlgAreaU`** *resize* method. If you use any of these methods, the dialog controls will not be automatically resized and positioned.

Table 6.4. DlgAreaU Non-Automatic Resize Controls

createCheckBoxGroup	createCheckBoxStem	createComboBoxInput	createEditInput
createEditInputGroup	createEditInputStem	createOkCancelLeftBottom	createOkCancelLeftTop
createOkCancelRightBottom	createOkCancelRightTop	createPushButtonGroup	createRadioButtonGroup
createRadioButtonStem			

All of these dialog controls can be added to the dialog template by using combinations of the methods recognized in the *resize* method.

6.2.12. Possible Problems

The **`DlgAreaU`** *resize* method can create slightly over-size margins on the left & bottom of the Dialog. To correct for this the DlgAreaU class has a *correctionFactor* attribute set by default to 1.05. In tests, this correction factor appears to neutralise the effect. If your dialogs have over (or under) sized margins, you may be able to correct this in your code by adjusting the *correctionFactor* attribute. For example:

```
u = .DlgAreaU~new(self)
u~correctionFactor=1.07
```

You will have to experiment to find the appropriate setting for this attribute.

This problem occurs because of the use of the *inaccurate* *factorX* and *factorY* attributes to convert pixels to dialog units. Fixing the problem, at this point, would most likely break existing programs.

6.2.13. Sample Code

```
/* DlgAreaDemo.Rex  --  Demonstrate DlgArea & DlgAreaU Classes  --  Feb 2006 */
```

```
dlg = .MyDialog~new
if dlg~initCode == 0 then dlg~execute("ShowTop")

return dlg~initCode

::requires "ooDialog.cls"
/* ========================================================================= */
::class 'MyDialog' subclass UserDialog
/* ========================================================================= */
::method init
/* ------------------------------------------------------------------------- */
  self~init:super
  if \ self~createCenter(250, 250, "MyDialog", "ThickFrame", , "Tahoma", 8) then do
    self~initCode = 1
    return
  end

  self~connectResize("onResize")

/* ------------------------------------------------------------------------- */
::method defineDialog
/* ------------------------------------------------------------------------- */
expose u

u = .dlgAreaU~new(self)                                       /* whole dlg   */
if u~lastError \= .nil then call errorDialog u~lastError
e = .dlgArea~new(u~x       ,u~y       ,u~w("70%"),u~h("90%")) /* edit   area */
s = .dlgArea~new(u~x       ,u~y("90%"),u~w("70%"),u~hr      ) /* status area */
b = .dlgArea~new(u~x("70%"),u~y       ,u~wr      ,u~hr      ) /* button area */

self~createEdit(IDC_EDIT, e~x, e~y, e~w, e~h, "multiline", "text")
self~createStaticText(IDC_ST_STATUS, s~x, s~y, s~w, s~h, , "Status info appears here")

self~createPushButton(IDC_PB_0, b~x, b~y('00%'), b~w, b~h('9%'), ,'Button' 0, 'Button'||0)
self~createPushButton(IDC_PB_1, b~x, b~y('10%'), b~w, b~h('9%'), ,'Button' 1, 'Button'||1)
self~createPushButton(IDC_PB_2, b~x, b~y('20%'), b~w, b~h('9%'), ,'Button' 2, 'Button'||2)
self~createPushButton(IDC_PB_3, b~x, b~y('30%'), b~w, b~h('9%'), ,'Button' 3, 'Button'||3)
self~createPushButton(IDC_PB_4, b~x, b~y('40%'), b~w, b~h('9%'), ,'Button' 4, 'Button'||4)
self~createPushButton(IDC_PB_5, b~x, b~y('50%'), b~w, b~h('9%'), ,'Button' 5, 'Button'||5)
self~createPushButton(IDC_PB_6, b~x, b~y('60%'), b~w, b~h('9%'), ,'Button' 6, 'Button'||6)
self~createPushButton(IDOK,     b~x, b~y('90%'), b~w, b~h('9%'),'DEFAULT', 'Ok')


/* ------------------------------------------------------------------------- */
::method unknown
/* ------------------------------------------------------------------------- */
use arg msgname, args
if msgname~abbrev("BUTTON") then
   self~newStatic(IDC_ST_STATUS)~setText('You Pressed Button' msgname~right(1))

/* ------------------------------------------------------------------------- */
::method onResize
/* ------------------------------------------------------------------------- */
expose u
use arg sizeEvent, sizeinfo
u~resize(self, sizeinfo)
```

This achieves the same dialog as the previous DlgArea *example*, but now it is resizable by dragging the frame.

6.3. ResizingAdmin Mixin Class

The **`ResizingAdmin`** class is a **`mixin`** class. Resizable dialogs are produced by having a dialog class inherit the **`ResizingAdmin`** class. Any ooDialog dialog that inherits the resizing admin class becomes resizable, using a default resizing policy, without adding any other code. Existing programs can add resizability to the dialogs in the programm by adding:

```
inherit ResizingAdmin
```

to the class definition of the dialog subclasses without making any other code changes. In many cases the default sizing policy produces an adequate resizable dialog. If not, methods of the **`ResizingAdmin`** can be used to specify a different default sizing policy, or to specify different resizing rules for individual controls.

6.3.1. Understanding How Resizing Works

The basic idea behind the **`ResizingAdmin`** class is relatively simple. The idea is that each edge, or side, of a dialog control window is *pinned* to an edge of the dialog window, or an edge of a dialog control within the dialog. As the edge of the *pinned to* window changes position, the *pinned* edge of the dialog control changes position according to the *type* of pin.

This can be visualized using a list-view in a dialog, where the left edge of the list-view is *pinned* to the left edge of the dialog and the right edge of the list-view is *pinned* to the right edge of the dialog. As the left edge of the dialog is pulled to the left, the pinned left edge of the list-view is pulled to the left along with it. The right edge of the list-view is pinned to the right edge of the dialog, so it remains stationary, (the right edge of the dialog is stationary.) As the dialog increases in width, the list-view increases in width also.

Every edge of every control in the dialog has a *sizing defintion* assigned to it. If the programmer does not explicitly define a sizing for an edge of a control, a default sizing definition is assigned to it by the ooDialog framework. This is how a dialog can be made resizable by just inheriting the **`ResizingAdmin`** and nothing else. The ooDialog framework will assign a default sizing definition to each edge of every control in the dialog.

Default Sizing Policy:

The initial default sizing policy is relatively simple. The left edge of a dialog control is pinned proportionally to the left edge of the dialog. The top edge of a dialog control is pinned proportionally to the top edge of the dialog. And so on, the right edge of a control to the right edge of the dialog, bottom edge of a control to the bottom edge of the dialog.

The default sizing policy can be changed by the programmer though the *defaultSizing* method. Or the individual default sizing for a single edge can be changed through the *defaultBottom*, *defaultLeft*, *defaultRight*, or *defaultTop* methods.

The default sizing definition may be adequate for the programmer's needs. If not, the programmer can explicitly set the sizing definitions for the edges of the controls as needed.

The sizing definitions for every control are placed in a table. When a resizing event happens, every control is resized, according to its definition, in the order the defintions occur in the table. Control sizings explicitly defined by the programmer are placed in the table in the order they are defined. All default control sizings generated by the ooDialog framework are placed in the table following the explicitly defined sizings, in the order the controls occur in the dialog template.

If an edge of a control is *pinned* to another window, the other window has to be resized first. This implies that the sizing defintion for a *pinned to* window has to precede the sizing definition for the window that is pinning to it.

Each sizing definition is comprised of 5 parts:

- The dialog control whose sizing is being defined. Dialog controls are specified by their *resource* ID. As in other areas of ooDialog, the ID may be numeric or *symbolic*.
- The *pinned edge* of the dialog control that is being defined. The pinned edge being defined is specified by one of four keywords: LEFT, TOP, RIGHT, and BOTTOM. Case is not significant.
- The *pinned to edge* of another window. *Pinned to* edges are specified by one of six keywords: LEFT, TOP, RIGHT, BOTTOM, XCENTER, and YCENTER. Case is not significant.
- The *pin type*. *Pin types* are specified by one of four keywords: PROPORTIONAL, STATIONARY, MYTOP, MYLEFT. Case is not significant.
- The *pinned to window*. The window an edge is pinned to must be either another dialog control window or the dialog window itself. Resource IDs are used to specify other dialog control windows. A special constant value, IDC_DEFAULT_PINTO_WINDOW, is provided to specify the dialog window. However, the dialog window is also the default, usually the IDC_IDC_DEFAULT_PINTO_WINDOW constant does not need to be specified.

Pinned edge:

The following specifies the meaning of the pinned edge keywords:

Left:

This should be self-explanatory, but technically it is the vertical edge of the window with the smallest X coordinate.

Top:

This should be self-explanatory, but technically it is the horizontal edge of the window with the smallest Y coordinate.

Right:

This should be self-explanatory, but technically it is the vertical edge of the window with the largest X coordinate.

Bottom:

This should be self-explanatory, but technically it is the horizontal edge of the window with the largest Y coordinate.

Pinned to edge:

The following specifies the meaning of the pinned to edge keywords:

Left:

This should be self-explanatory, but technically it is the vertical edge of the window with the smallest X coordinate.

Top:

This should be self-explanatory, but technically it is the horizontal edge of the window with the smallest Y coordinate.

Right:

This should be self-explanatory, but technically it is the vertical edge of the window with the largest X coordinate.

Bottom:

This should be self-explanatory, but technically it is the horizontal edge of the window with the largest Y coordinate.

XCenter:

The edge being pinned keeps its relationship to the horizontal center of the XCenter pinned to edge. This works best when the initial position of the control is already horizontally centered with the pinned to window. Only the top and bottom edges of a dialog control can be pinned to a XCenter edge.

YCenter:

The edge being pinned keeps its relationship to the vertical center of the YCenter pinned to edge. This works best when the initial position of the control is already vertically centered with the pinned to window. Only the left and right edges of a dialog control can be pinned to a YCenter edge.

Pin type:

The following specifies the meaning of the pin type keywords:

Stationary

The edge being pinned remains a fixed distance from the pin to edge. The fixed distance is the distance in pixels between the pinned and pinned to edges when the dialog is initially created. For instance, if the left side of a list-view is pinned to the left side of the dialog, and the list-view is initially 7 pixels from the left edge of the dialog, the left edge of the list-view will remain 7 pixels from the left edge of the dialog no matter what size the dialog is resized to.

Proportional:

The edge being pinned remains a proportional distance from the pin to edge. Again, the proportion depends on the initial distance between the two edges when the dialog is first created. Using the example of a list-view whose left edge is 7 pixels from the left edge of the dialog. If the left edge of the list-view is pinned to the left edge of the dialog using a proportional pin, if the dialog is doubled in width, the list-view will end up 14 pixels from the left edge of the dialog.

MyLeft:

A MyLeft pin causes the width of the dialog control to stay constant. The initial width of the control does not change as the dialog is resized. The MyLeft type of pin can only be used for the right edge of a dialog control. The pin to window and pin to edge are ignored with this type of pin, but most methods require the pin to edge argument. Typically a line of code for this type of pin would look like this:

```
self~controlRight(IDC_ST_PROCESSA, 'MYLEFT', 'LEFT')
```

The *LEFT* argument in the above is actually ignored. IDC_ST_PROCESSA is a valid symbolic ID in the application.

MyTop:

A MyTop pin is the converse of a MyLeft pin. It causes the height of the dialog control to stay constant. The initial height of the control does not change as the dialog is resized. The MyTop type of pin can only be used for the bottom edge of a dialog control. The pin to window and pin to edge are ignored with this type of pin, but most methods require the pin to edge argument. Typically a line of code for a MyTop pin would look like this:

```
self~controlBottom(IDC_ST_PROCESSB, 'MYTOP', 'TOP')
```

The *TOP* argument in the above is actually ignored. IDC_ST_PROCESSB is a valid symbolic ID in the application.

There are a number of sizing definitons that would result in incorrectly sized controls. Take for instance control A that is pinned to an edge in control B. If control B were to come after control A in the sizing table, then when control A was being resized, it would be resized in relationship to the *old* size of control B. Whatever relationship was desired with the control B window would not be carried out correctly. In a case like this, the Rexx programmer would see that the dialog was not resized the way intended, but it would be very difficult to determine why. Because of this, when this documentation says things like: *the pin to window B must precede the window being pinned in the sizing table*, what is really meant is that a syntax condition is raised if that condition is detected.

Note, when using the **`ResizingAdmin`** class, the event notifications related to resizing are handled internally by the ooDialog framework and the event notifications can not be connected to an event handler in the Rexx dialog. There are 3 event notifications effected: the SIZE, SIZING, and EXITSIZEMOVE events. The *connectResize*, *connectResizing*, and *connectSizeMoveEnded* methods have no effect when a dialog has inherited the **`ResizingAdmin`** class. However, the *wantSizeEnded* can be used to allow the application to be notified of the EXITSIZEMOVE event.

6.3.2. Method Table

The following table lists the methods of the **`ResizingAdmin`** class for use by the programmer:

Table 6.5. ResizingAdmin Methods

Method	Description
Constant Methods	**Constant Methods**
IDC_DEFAULT_PINTO_WINDOW	The symbolic ID of the default window an edge of a dialog control is pinned to. This ID resolves to the dialog window.
Attribute Methods	**Attribute Methods**
maxSize	Reflects the maximum size the dialog is allowed to resize to. By default, there is no limit on the maximum size for the dialog.
minSize	Reflects the minimum size the dialog is allowed to resize to. By default, the minimum size for the dialog is set to its initial size when the operating system first creates it.
Instance Methods	**Instance Methods**
controlBottom	Defines the sizing for the bottom edge of the dialog control specified.
controlLeft	Defines the sizing for the left edge of the dialog control specified.
controlRight	Defines the sizing for the right edge of the dialog control specified.
controlSizing	Defines the sizing for some or all of the edges of the specified dialog control with one method call.
controlTop	Defines the sizing for the top edge of the dialog control specified.
defaultBottom	Changes the default sizing policy for the bottom edge of dialog controls.
defaultLeft	Changes the default sizing policy for the left edge of dialog controls.

Method	Description
defaultRight	Changes the default sizing policy for the right edge of dialog controls.
defaultSizing	Resets the default sizing policy to that specified.
defaultTop	Changes the default sizing policy for the top edge of dialog controls.
defineSizing	The *defineSizing* method is invoked automatically by the ooDialog framework. It is not meant to be, nor should it be, invoked by the programmer. The method is meant to be over-ridden by the programmer. It is used to add or change sizing defintions.
noMaxSize	Removes any exsiting restriction on the maximum size the dialog can be resized to.
noMinSize	Removes any exsiting restriction on the minimum size the dialog can be resized to.
pagedTab	Adds the data necessary to properly resize the dialogs used as pages in a tab control.
useDefaultSizing	Sets the sizing information for the specified control to that of the current *default* sizing.
wantSizeEnded	Allows the user to specify that a method in the Rexx dialog is invoked when the EXITSIZEMOVE event happens.

6.3.3. maxSize (Attribute)

```
>>--maxSize---------------------------------------><

>>--maxSize = sizeObj-----------------------------><
```

Reflects the maximum size the dialog is allowed to resize to. By default, there is no limit on the maximum size for the dialog.

maxSize get:

Returns the maximum size constraint of the dialog as a *Size* object. If the maximum size is not constrained, the **`.nil`** object is returned.

maxSize set:

Sets the maximum size a dialog can be sized to using a **`Size`** object. The *width* of the **`size`** object is the maximum width, in pixels, for the dialog and the *height* of the **`Size`** object is the maximum height, in pixels for the dialog. By default, the dialog will not have a maximum size restriction.

Remarks:

The *maxSize* attribute can only be set using a **`Size`** object. If a maximum size constraint has been set, and for some reason that constraint needs to be removed, use the *noMaxSize* method.

Details:

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. However, the *maxSize* attribute can be set or read at any time in the life-cycle of the dialog.

Example:

This example sets the maximum size for the resizable dialog based on a proportion of the initial dialog size.

```
::method initDialog

  size = self~MinSize
  max = .Size~new(size~width * 2, trunc(size~height * 1.5))
  self~maxSize = max
```

6.3.4. minSize (Attribute)

```
>>--minSize--------------------------------------><

>>--minSize = sizeObj----------------------------><
```

Reflects the minimum size the dialog is allowed to resize to. By default, the minimum size for the dialog is set to its initial size when the operating system first creates it.

minSize get:

Returns the minimum size constraint of the dialog as a *Size* object. If the minimum size is not constrained, the **`.nil`** object is returned.

minSize set:

Set the minimum size the dialog can be sized to using a **`Size`** object. The *width* of the **`size`** object is the minimum width, in pixels, for the dialog and the *height* of the **`Size`** object is the minimum height, in pixels, for the dialog. By default, the dialog will have an initial minimum size restriction set to the size of the dialog when first created.

Remarks:

The *minSize* attribute can only be set using a **`Size`** object. If a minimum size constraint is set, and that constraint needs to be removed, use the *noMinSize* method.

Details

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. However, the *minSize* attribute can be set or read at any time in the life-cycle of the dialog.

Example:

This example sets a minimum size for the resizable dialog to be 100 pixels wide by 50 pixels high:

```
::method defineSizing

  s = .Size~new(100, 50)
  self~minSize = s

  return 0
```

6.3.5. controlBottom

```
>>--controlBottom(--id--,--howPinned--,--pinToEdge--+------------------+--)----><
```

```
                                                   +--,--pinToWindow--+
```

Defines the sizing for the bottom edge of the dialog control specified.

Arguments:

The arguments are:

id [required]
: The resource ID of the dialog control whose sizing is being defined. May be numeric or *symbolic*.

howPinned [required]
: The *pin type* keyword for the sizing definition.

pinToEdge [required]
: The *pinned to edge* keyword for the sizing definition.

pinToWindow [optional]
: The window the bottom edge is being pinned to. By default this is the dialog window. To specify a dialog control window use the resource ID of the control. This may be numeric or *symbolic*. The *IDC_IDC_DEFAULT_PINTO_WINDOW* constant can also be used to explicitly define the pin to window as the dialog window.

Return value:

Returns 0 always.

Remarks:

When defining the sizing for the bottom edge, the definition **must** be correct. If an incorrect definition is detected, a syntax condition is raised.

Details

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. The *controlBottom* method is one of them. It can only be invoked during the *defineSizing* method.

Example:

This example defines the sizing for the bottom edge of a list-view:

```
self~controlBottom(IDC_LV_MAIN, 'STATIONARY', 'BOTTOM')
```

6.3.6. controlLeft

```
>>--controlLeft(--id--,--howPinned--,--pinToEdge--+------------------+--)------><
                                                  +--,--pinToWindow--+
```

Defines the sizing for the left edge of the dialog control specified.

Arguments:

The arguments are:

id [required]
: The resource ID of the dialog control whose sizing is being defined. May be numeric or *symbolic*.

howPinned [required]
: The *pin type* keyword for the sizing definition.

pinToEdge [required]
: The *pinned to edge* keyword for the sizing definition.

pinToWindow [optional]
: The window the left edge is being pinned to. By default this is the dialog window. To specify a dialog control window use the resource ID of the control. This may be numeric or *symbolic*. The *IDC_IDC_DEFAULT_PINTO_WINDOW* constant can also be used to explicitly define the pin to window as the dialog window.

Return value:

Returns 0 always.

Remarks:

When defining the sizing for the left edge, the definition **must** be correct. If an incorrect definition is detected, a syntax condition is raised.

Details

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. The *controlLeft* method is one of them. It can only be invoked during the *defineSizing* method.

Example:

This example defines the sizing for the left edge of a list-view:

```
self~controlLeft(IDC_LV_MAIN, 'STATIONARY', 'LEFT')
```

6.3.7. controlRight

```
>>--controlRight(--id--,--howPinned--,--pinToEdge--+------------------+--)-----><
                                                   +--,--pinToWindow--+
```

Defines the sizing for the right edge of the dialog control specified.

Arguments:

The arguments are:

id [required]
: The resource ID of the dialog control whose sizing is being defined. May be numeric or *symbolic*.

howPinned [required]
: The *pin type* keyword for the sizing definition.

pinToEdge [required]
: The *pinned to edge* keyword for the sizing definition.

pinToWindow [optional]
: The window the right edge is being pinned to. By default this is the dialog window. To specify a dialog control window use the resource ID of the control. This may be numeric or *symbolic*.

The *IDC_IDC_DEFAULT_PINTO_WINDOW* constant can also be used to explicitly define the pin to window as the dialog window.

Return value:

Returns 0 always.

Remarks:

When defining the sizing for the right edge, the definition **must** be correct. If an incorrect definition is detected, a syntax condition is raised.

Details

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. The *controlRight* method is one of them. It can only be invoked during the *defineSizing* method.

Example:

This example defines the sizing for the right edge of a list-view. To be sure the code is clear to anyone looking at it, the programmer decided to explicitly name the window the right edge is pinned to:

```
 self~controlRight(IDC_LV_MAIN, 'STATIONARY', 'RIGHT',
self~IDC_IDC_DEFAULT_PINTO_WINDOW)
```

6.3.8. controlSizing

```
>>--controlSizing(--id--+--------- +--+--------+--+---------+--+-----------+--)-----><
                        +-,-right--+  +-,-top--+  +-,-left--+  +-,-bottom--+
```

Defines the sizing for some or all of the edges of the specified dialog control with one method call.

Arguments:

The arguments are:

id [required]

The resource ID of the dialog control whose sizing is being defined. May be numeric or *symbolic*.

right [optional]

An array whose indexes define the sizing for the right edge of the control. If this argument is omitted, the current default *right* sizing definition is used. See the remarks section for the details on constructing the array and its indexes.

top [optional]

An array whose indexes define the sizing for the top edge of the control. If this argument is omitted, the current default *top* sizing definition is used. See the remarks section for the details on constructing the array and its indexes.

left [optional]

An array whose indexes define the sizing for the left edge of the control. If this argument is omitted, the current default *left* sizing definition is used. See the remarks section for the details on constructing the array and its indexes.

bottom [optional]
: An array whose indexes define the sizing for the bottom edge of the control. If this argument is omitted, the current default *bottom* sizing definition is used. See the remarks section for the details on constructing the array and its indexes.

Return value:

Returns 0 on success, 1 on error.

Remarks:

The *right*, *top*, etc., arguments must be an array. Each array must contain index 1 and index 2. Index 3 is optional. The arrays are formed are as follows:

Index 1 [required]
: The *pin type* keyword for the sizing definition of the edge.

Index 2 [required]
: The *pinned to edge* keyword for the sizing definition of the edge.

Index 3 [optional]
: The window the edge is being pinned to. By default this is the dialog window. To specify a dialog control window use the resource ID of the control. This may be numeric or *symbolic*. The *IDC_IDC_DEFAULT_PINTO_WINDOW* constant can also be used to explicitly define the pin to window as the dialog window.

For any of the optional edge sizing definition arguments that are omitted, a sizing edge defintion is constructed using the default sizing in effect at the time. Not that the default sizing can be changed using the *defaultLeft*, *defaultTop*, etc., methods. Changing the default sizing has no effect on the sizing definition for any control that has already been added to the sizing table. It only affects sizing added after the default has been changed.

Details

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. The *controlSizing* method is one of them. It can only be invoked during the *defineSizing* method.

Example:

This example defines the sizing for all edges of an edit control using the *controlSizing* method. Note that there are any number of ways the Rexx code could be formatted here. The format shown is merely one way, it is not required:

```
self~controlSizing(IDC_EDIT_NAMES,                                    -
                   .array~of('STATIONARY', 'LEFT', IDC_ST_NAMES),     -
                   .array~of('STATIONARY', 'TOP',  IDC_GB_TEST),      -
                   .array~of('STATIONARY', 'LEFT', IDC_ST_LABELS),    -
                   .array~of('MYTOP',      'TOP')                     -
                   )
```

6.3.9. controlTop

```
>>--controlTop(--id--,--howPinned--,--pinToEdge--+------------------+--)------><
                                                 +--,--pinToWindow--+
```

Defines the sizing for the bottom edge of the dialog control specified.

Arguments:

The arguments are:

id [required]
: The resource ID of the dialog control whose sizing is being defined. May be numeric or *symbolic*.

howPinned [required]
: The *pin type* keyword for the sizing definition.

pinToEdge [required]
: The *pinned to edge* keyword for the sizing definition.

pinToWindow [optional]
: The window the top edge is being pinned to. By default this is the dialog window. To specify a dialog control window use the resource ID of the control. This may be numeric or *symbolic*. The *IDC_IDC_DEFAULT_PINTO_WINDOW* constant can also be used to explicitly define the pin to window as the dialog window.

Return value:

Returns 0 always.

Remarks:

When defining the sizing for the top edge, the definition **must** be correct. If an incorrect definition is detected, a syntax condition is raised.

Details

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. The *controlTop* method is one of them. It can only be invoked during the *defineSizing* method.

Example:

This example defines the sizing for the top edge of a list-view:

```
self~controlTop(IDC_LV_MAIN,'STATIONARY', 'TOP')
```

6.3.10. defaultBottom

```
>>--defaultBottom(--howPinned--,--pinToEdge--+------------------+--)----------><
                                             +--,--pinToWindow--+
```

Changes the default sizing policy for the bottom edge of dialog controls. Before the underlying Windows dialog starts executing, every edge of every dialog control in the dialog is assigned a sizing definition. Every edge that the application has not explicitly defined a sizing for is assigned the default sizing definition for that edge.

Arguments:

The arguments are:

howPinned [required]
: The *pin type* keyword for the sizing definition.

pinToEdge [required]

The *pinned to edge* keyword for the sizing definition.

pinToWindow [optional]

The window the bottom edge is being pinned to. By default this is the dialog window. To specify a dialog control window use the resource ID of the control. This may be numeric or *symbolic*. The *IDC_IDC_DEFAULT_PINTO_WINDOW* constant can also be used to explicitly define the pin to window as the dialog window.

Return value:

Returns 0 on success, 1 on error.

Remarks:

If the *pinToWindow* argument is used and its value is not the dialog window, then a sizing record for that window must already exist in the sizing table. That is, if the *pinToWindow* is set to dialog control AA, then a sizing definition for AA must have been already explicitly defined.

Details

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. The *defaultBottom* method is one of them. It can only be invoked during the *defineSizing* method.

Example:

This example ...

```

```

6.3.11. defaultLeft

```
>>--defaultLeft(--howPinned--,--pinToEdge--+------------------+--)-------------><
                                            +--,--pinToWindow--+
```

Changes the default sizing policy for the left edge of dialog controls. Before the underlying Windows dialog starts executing, every edge of every dialog control in the dialog is assigned a sizing definition. Every edge that the application has not explicitly defined a sizing for is assigned the default sizing definition for that edge.

Arguments:

The arguments are:

howPinned [required]

The *pin type* keyword for the sizing definition.

pinToEdge [required]

The *pinned to edge* keyword for the sizing definition.

pinToWindow [optional]

The window the left edge is being pinned to. By default this is the dialog window. To specify a dialog control window use the resource ID of the control. This may be numeric or *symbolic*. The *IDC_IDC_DEFAULT_PINTO_WINDOW* constant can also be used to explicitly define the pin to window as the dialog window.

Return value:

Returns 0 on success, 1 on error.

Remarks:

If the *pinToWindow* argument is used and its value is not the dialog window, then a sizing record for that window must already exist in the sizing table. That is, if the *pinToWindow* is set to dialog control AA, then a sizing definition for AA must have been already explicitly added to the sizing table.

Details

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. The *defaultLeft* method is one of them. It can only be invoked during the *defineSizing* method.

Example:

This example ...

```

```

6.3.12. defaultRight

```
>>--defaultRight(--howPinned--,--pinToEdge--+------------------+--)-----------><
                                            +--,--pinToWindow--+
```

Changes the default sizing policy for the right edge of dialog controls. Before the underlying Windows dialog starts executing, every edge of every dialog control in the dialog is assigned a sizing definition. Every edge that the application has not explicitly defined a sizing for is assigned the default sizing definition for that edge.

Arguments:

The arguments are:

howPinned [required]

The *pin type* keyword for the sizing definition.

pinToEdge [required]

The *pinned to edge* keyword for the sizing definition.

pinToWindow [optional]

The window the right edge is being pinned to. By default this is the dialog window. To specify a dialog control window use the resource ID of the control. This may be numeric or *symbolic*. The *IDC_IDC_DEFAULT_PINTO_WINDOW* constant can also be used to explicitly define the pin to window as the dialog window.

Return value:

Returns 0 on success, 1 on error.

Remarks:

If the *pinToWindow* argument is used and its value is not the dialog window, then a sizing record for that window must already exist in the sizing table. That is, if the *pinToWindow* is set to dialog control AA, then a sizing definition for AA must have been already explicitly added to the sizing table.

Details

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. The *defaultRight* method is one of them. It can only be invoked during the *defineSizing* method.

Example:

This example ...

```

```

6.3.13. defaultSizing

```
>>--defaultSizing(--+---------+--+---------+--+----------+--+------------+--)--><
                    +--right--+  +-,--top--+  +-,--left--+  +-,--bottom--+
```

Resets the default sizing policy. At the time the underlying dialog starts executing, every edge of every control in the dialog must have a sizing definition. Any edge of any control that does not have a sizing definition explicitly set by the application is assigned the default sizing definition for that edge.

Arguments:

The arguments are:

right [optional]
: An array whose indexes define the default sizing for the right edge of a control. If this argument is omitted, the current default right sizing definition is not changed. See the remarks section for the details on constructing the array and its indexes.

top [optional]
: An array whose indexes define the default sizing for the top edge of a control. If this argument is omitted, the current default top sizing definition is not changed. See the remarks section for the details on constructing the array and its indexes.

left [optional]
: An array whose indexes define the default sizing for the left edge of a control. If this argument is omitted, the current default left sizing definition is not changed. See the remarks section for the details on constructing the array and its indexes.

bottom [optional]
: An array whose indexes define the sizing for the bottom edge of the control. If this argument is omitted, the current default bottom sizing definition is not changed. See the remarks section for the details on constructing the array and its indexes.

Return value:

Returns 0 on success and 1 on error.

Remarks:

The *right*, *top*, etc., arguments must be an array. Each array must contain index 1 and index 2. Index 3 is optional. The arrays are formed are as follows:

Index 1 [required]
: The *pin type* keyword for the sizing definition of the edge.

Index 2 [required]

The *pinned to edge* keyword for the sizing definition of the edge.

Index 3 [optional]

The window the edge is being pinned to. By default this is the dialog window. To specify a dialog control window use the resource ID of the control. This may be numeric or *symbolic*. The *IDC_IDC_DEFAULT_PINTO_WINDOW* constant can also be used to explicitly define the pin to window as the dialog window.

For any of the optional edge sizing definition arguments that are omitted, the current sizing edge defintion is left unchanged. Note that the default sizing for a single edge can be changed individually using the *defaultLeft*, *defaultTop*, etc., methods. Changing the default sizing has no effect on the sizing defintion for any control that has already been added to the sizing table. It only effects sizing definitions added after the default has been changed.

The default sizing definition is used for any control that does not have any sizing record in the sizing table. It is also used for any missing edge defintions in a partially defined record. E.g., if the application explicitly sets only the top edge definition for a dialog control, the default sizing definitions for the left, right, and bottom edges will be used.

Details

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. The *defaultSizing* method is one of them. It can only be invoked during the *defineSizing* method.

Example:

This example comes from an application that changes the default sizing to one that is better for the application. All the dialog controls whose sizing is not explicitly defined in the application will have their left and right sides pinned to the left side of dialog. The STATIONARY keyword means that the sides will remain the same pixel distance from the left edge as they start with. A similar thing with the top and bottom of the controls, they will be pinned to the bottom of a list-view. The effect of this, is that the dialog controls using the default sizing will all remain the same size. They will not get bigger or smaller:

```
self~defaultSizing(.array~of('STATIONARY', 'LEFT'),                    -
                   .array~of('STATIONARY', 'BOTTOM', IDC_LV_MAIN),     -
                   .array~of('STATIONARY', 'LEFT'),                    -
                   .array~of('STATIONARY', 'BOTTOM', ICD_LV_MAIN)      -
                  )
```

6.3.14. defaultTop

```
>>--defaultTop(--howPinned--,--pinToEdge--+------------------+--)-------------><
                                          +--,--pinToWindow--+
```

Changes the default sizing policy for the top edge of dialog controls. Before the underlying Windows dialog starts executing, every edge of every dialog control in the dialog is assigned a sizing definition. Every edge that the application has not explicitly defined a sizing for is assigned the default sizing definition for that edge.

Arguments:

The arguments are:

howPinned [required]
: The *pin type* keyword for the sizing definition.

pinToEdge [required]
: The *pinned to edge* keyword for the sizing definition.

pinToWindow [optional]
: The window the top edge is being pinned to. By default this is the dialog window. To specify a dialog control window use the resource ID of the control. This may be numeric or *symbolic*. The *IDC_IDC_DEFAULT_PINTO_WINDOW* constant can also be used to explicitly define the pin to window as the dialog window.

Return value:

Returns 0 on success, 1 on error.

Remarks:

If the *pinToWindow* argument is used and its value is not the dialog window, then a sizing record for that window must already exist in the sizing table. That is, if the *pinToWindow* is set to dialog control AA, then a sizing definition for AA must have been already explicitly added to the sizing table.

Details

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. The *defaultTop* method is one of them. It can only be invoked during the *defineSizing* method.

Example:

This example changes the default sizing for dialog control top edges to a STATIONARY type of pin and pins the edges to the bottom edge of a list-view:

```
self~defaultTop~('STATIONARY', 'BOTTOM', IDC_LV_MAIN)
```

6.3.15. defineSizing

```
>>--defineSizing---------------------------------><
```

The *defineSizing* method is invoked automatically by the ooDialog framework. It is not meant to be, nor should it be, invoked by the programmer. The method is meant to be over-ridden by the programmer. It is used to add or change sizing defintions.

Arguments:

There are no arguments sent to the *defineSizing* method.

Return value:

Returns 0, always.

Remarks:

Many of the methods of the **`ResizingAdmin`** can only be invoked during the execution of the *defineSizing* method. All sizing definitions must be done within the *defineSizing* method. The *defineSizing* method executes before the underlying Windows dialog is created. The implication of this is that methods that require the underlying Windows dialog to exist can not be used in *defineSizing*.

The *defineSizing* method is invoked as the last procedure in the initialization of the base dialog object. Error conditions during the *defineSizing* method cause the initialization of the base dialog object to fail.

Note, the default implementation of the *defineSizing* method returns 0. If an application over-rides the *defineSizing* method, that method **must** also return 0.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows the complete *defineSizing* method from one of the example programs:

```
::method defineSizing

    -- The only control in this dialog is the list-view.  We define its sizing
    -- so that each edge of the list-view maintains the same distance to its
    -- corresponding edge of the dialog:
    self~controlSizing(IDC_LV_MAIN,                                   -
                       .array~of('STATIONARY', 'LEFT'),               -
                       .array~of('STATIONARY', 'TOP'),                -
                       .array~of('STATIONARY', 'RIGHT'),              -
                       .array~of('STATIONARY', 'BOTTOM')              -
                      )

    -- A value must be returned from the defineSizing() method.  0 allows the
    -- dialog to continue, any other value is a failure and the dialog will be
    -- ended.
    return 0
```

6.3.16. noMaxSize

```
>>--noMaxSize------------------------------------><
```

Removes any exsiting restriction on the maximum size the dialog can be resized to.

Arguments:

There are no arguments for this method.

Return value:

Returns 0, always.

Remarks:

By default no maximum size limit is set for resizable dialogs. If the application uses the *maxSize* attribute to set a maximum size and later needs to remove the restriction, it can be done using the *noMaxSize* method.

Details

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. The *noMaxSize* method is not one of them. It can be invoked any time during the life-cycle of the dialog.

6.3.17. noMinSize

```
>>--noMinSize------------------------------------><
```

Removes any exsiting restriction on the minimum size the dialog can be resized to.

Arguments:

This method takes no arguments.

Return value:

Returns 0, always.

Remarks:

By default the minimum size limit is set to the initial size of the dialogs. If the application wants to remove this restriction, or uses the *minSize* attribute to set a minimum size and later needs to remove the restriction, it can be done using the *noMinSize* method.

Details

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. The *noMinSize* method is not one of them. It can be invoked any time during the life-cycle of the dialog.

Example:

This example removes the minimum size restriction for a resizable dialog. It does it during the *defineSizing* method because it is convenient to do it there, not because it is required to be done in that method:

```
::method defineSizing

  self~noMinSize

  return 0
```

6.3.18. pagedTab

```
>>--pagedTab(--tabID--,--dlgIDs--)---------------><
```

Adds the data necessary to properly resize the dialogs used as pages in a tab control.

Arguments:

The arguments are:

tabID [required]

The resource ID of the tab control that contains the dialogs used as pages of that tab control. May be numeric or *symbolic*.

dlgIDs [required]

An array of the dialog *IDs* of each control dialog. Each item in the array is assumed to be the dlgID of one of the dialog pages. Each item in the array may be numeric or symbolic. The items must be valid dialog IDs, that is a whole number greater than 0, or a symbolic ID that resolves to a whole number greater than 0.

The array must not be sparse and the number of items can not exceed the maximum number of child dialogs, which is 20. The order of the IDs does not matter.

Return value:

Returns 0 on success. On any detected error a syntax condition is raised.

Remarks:

A *paged* tab is a tab control that uses *ControlDialog* dialogs as the contents for its pages. A resizable dialog can only define up to 4 paged tabs.

In order for the resizing admin to properly resize **`ControlDialog`** dialogs embedded in a tab control, the *pagedTab* method must be used to define each paged tab.

Details

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. The *pagedTab* method is one of them. It can only be invoked during the *defineSizing* method.

Example:

This example shows part of the *defineSizing* method for a dialog that has a tab control with 5 pages:

```
dlgIDs = .array~of(IDD_LISTVIEW_DLG, IDD_TREEVIEW_DLG, IDD_PROGRESSBAR_DLG, -
                   IDD_TRACKBAR_DLG, IDD_TAB_DLG)

self~pagedTab(IDC_TAB, dlgIDs)
```

6.3.19. useDefaultSizing

```
>>--useDefaultSizing(--ctrlID--)-----------------><
```

Sets the sizing information for the specified control to that of the current *default* sizing.

Arguments:

The single argument is:

ctrlID

The resource ID of the control whose sizing is being set. May be numeric or *symbolic*.

Return value:

Returns 0 on success, 1 on error.

Remarks:

In order to pin the edge of a dialog control to the edge of some other dialog control, a sizing definition for the other control must already exist in the sizing definition table. The *useDefaultSizing* is a convenience method that allows a sizing definition for the *other* control to be placed in the table with a minimum of effort.

Details

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. The *useDefaultSizing* method is one of them. It can only be invoked during the *defineSizing* method.

Example:

In this example the default sizing is going to be changed to pin the top and bottom edges of controls to a list-view in the application. There needs to be a sizing definition for the list-view already in the table to do that, but the sizing for the list-view is that of the current default sizing. The *useDefaultSizing* method provides a convenient way to add the list-view's record:

```
self~useDefaultSizing(IDC_LV_MAIN)

self~defaultSizing(.array~of('STATIONARY', 'LEFT'),                        -
                   .array~of('STATIONARY', 'BOTTOM', IDC_LV_MAIN),         -
                   .array~of('STATIONARY', 'LEFT'),                        -
                   .array~of('STATIONARY', 'BOTTOM', ICD_LV_MAIN)          -
                  )
```

6.3.20. wantSizeEnded

```
>>--wantSizeEnded(--+-----------+--+----------------+--)--------><
                    +--mthName--+  +--,--willReply--+
```

Allows the user to specify that a method in the Rexx dialog is invoked when the EXITSIZEMOVE event happens.

Arguments:

The arguments are:

mthName [optional]

The name of the event handler method in the Rexx dialog to be connected to the EXITSIZEMOVE event. If this argument is omitted, the ooDialog framework uses a method name of *onSizeEnded*.

willReply [optional]

Specifies if the interpreter should *wait*, or not wait, for the reply from the event handler. If *willReply* is true, the interpreter waits in the window message processing loop until the event handler returns a reply to the notification, if false the interpreter does not wait.

Return value:

Returns 0, always.

Remarks:

When using the **`ResizingAdmin`** class, the event notifications related to resizing are handled internally by the ooDialog framework and the event notifications can not be connected to an event handler in the Rexx dialog. There are 3 event notifications effected: the SIZE, SIZING, and EXITSIZEMOVE events. The *connectResize*, *connectResizing*, and *connectSizeMoveEnded* methods have no effect when a dialog has inherited the **`ResizingAdmin`** class.

However, there are cases when the resize ended notification may be needed by an application with a resizable dialog. The *wantSizeEnded* method is for those cases.

Details

Raises syntax errors when incorrect usage is detected.

Some methods of the *ResizingAdmin* can only be invoked during the *defineSizing* method. The *wantSizeEnded* method is not one of them. It can be invoked any time during the life-cycle of the dialog.

Example:

This example informs the ooDialog framework that the dialog needs the EXITSIZEMOVE event notification. It explicity specifies the name of the event handling method, even though the method name is the same as the default:

```
self~wantSizeEnded('onSizeEnded', .true)
```

6.3.20.1. ExitSizeMove Event Handler

The event handler for the exit size move event is invoked when the user has been dragging the sizing border of a resizable dialog and releases the mouse button.

Whether the event handler must return a value and if the interpreter waits (or does not wait) for this return is specified by using the *willReply* argument in the *wantSizeEnded* method. When *willReply* is true, the event handler must return a value and the interpreter waits for that return. Otherwise, the interpreter does not wait for the return and the event handler is not *required* to return a value. Good practice would be to always return a value from an event handler.

```
::method onSizeEnded unguarded
  use arg

  return 0
```

Arguments:

The event handling method does not receive any arguments.

Return:

The operating system ignores the return for this event so any value can be returned. 0 makes a good return value.

Example

The following example comes from an application that uses *ControlDialog* dialogs as the pages in a *Tab* control. When the resizing of the dialog ends, the dialog used for page 5 needs to do some extra processing, which is done in the placeButton() method:

```
::method onSizeEnded unguarded
    expose tabContent needCalculation

    needCalculation = .true

    tabContent[5]~placeButton

    return 0
```

Resource File Dialogs

The resource definition, or dialog *template*, for a dialog can either be constructed manually, (see the *UserDialog* Class,) or read from a file. ooDialog provides two main dialog classes that read the dialog templates from a file: the *ResDialog* class and the *RcDialog* class.

The **`ResDialog`** class uses a dialog template from a binary file and a **`RcDialog`** class gets the template from a resource script file. Resource script files are plain text files. In general, both types of files are produced by resource editors which allow the user to visually design the dialog in a "What You See Is What You Get" manner. Since resource definitions are standardized, resource editors that produce Windows compatible files can be used to define dialogs for use with ooDialog.

7.1. ResDialog Class

The ResDialog class is designed to be used together with a binary (compiled) resource. Compiled resources can be attached to a *module*, which in Windows is a binary executable file. (Usually, a file with the extension .exe or .dll). It is possible to create a DLL that has no executable code and attach compiled resources to it. The common term for this type of DLL is a *resource only* DLL. A resource only DLL is the type of module that makes the most sense to use with the ResDialog class. However, there is no reason why any DLL, that has the compiled resource attached to it, could not be used.

Note: the **`ResDialog`** class is a concrete subclass of the *dialog* object and therefore has all of the methods of the dialog object. From the Rexx programmer's perspective, the only difference to be aware of is in the *new* method.

7.1.1. new (Class method)

```
>>--new(--module--,--id--+------------+--+----------+--)----------------------><
                         +-,--dlgData.-+  +-,--hFile-+
```

Instantiates a new **`ResDialog`** object. The dialog template for the object is taken from the specified module, which is usually a resource only DLL.

Arguments:

The arguments when creating a new dialog instance of this class are:

module [required]

The file name of the executable module (a DLL or EXE) in which the resource (the compiled dialog template) is located.

id [required]

The resource ID of the dialog template. This may be numeric or *symbolic*. The resource ID is assigned to the dialog template when it was compiled.

dlgData. [optional]

A *dialog data stem* variable (don't forget the trailing period) that contains data used to initialize the underlying dialog's controls.

hFile [optional]

The name of a *file* containing symbolic ID defines for resource IDs.

Remarks:

Normally a programmer does not instantiate a **`ResDialog`** directly, but rather creates a subclass of a **`ResDialog`** and instantiates the subclass. Recall that in ooRexx, when a new object is instantiated, the *new* method invokes the *init*() method of the object, using the method arguments of the *new*() method. So, the arguments of the *new* method are also the arguments of the *init* method.

Therefore, if the programmer over-rides the *init* method in the subclass of the **`ResDialog`**, care must be taken to invoke the superclass *init* method with the correct arguments.

When a **`ResDialog`**, or subclass, is instantiated, and the file specified in the *module* argument can not be found, or can not be loaded as an executable module, a message box will pop up informing the user of the problem.

Example:

This sample code creates a new dialog object from a subclass of **`ResDialog`**. It uses the dialog template with resource ID 100 in the **`MyDlg.dll`** file. The state of the dialog's controls are initialized with the values of the ctrlData. stem variable.

```
ctrlData.101="1"
ctrlData.102="Please enter your password."
ctrlData.103=""

dlg = .PasswordDialog~new("passwordDlg.dll", 100, ctrlData.)
if dlg~initCode <> 0 then do
  say 'The dialog could not be created'
  return .false
  /* Or some other error handling code. */
end
  ...

::requires "ooDialog.cls"

::class "PasswordDialog" subclass ResDialog

...
```

An additional *example* can be found under the *new* method of the dialog object that shows the use of the dialog data stem and header file arguments in more detail.

7.2. RcDialog Class

The **`RcDialog`** class gets its dialog *template* from a resource script file. Resource script files are plain text files usually produced by a resource editor. The files generally have a file extension of ".rc", but an extension of ".dlg" is used by some resource editors.

Notes:

The **`RcDialog`** class subclasses the *UserDialog* class. The **`UserDialog`** is a concrete subclass of the *dialog* object. Therefore the **`RcDialog`** has available all of the methods of the dialog object and the **`UserDialog`**. For the Rexx programmer, there are only a few differences to be aware of between the **`UserDialog`** and the **`RcDialog`**.

The **`RcDialog`** class is a convenience class that uses the *load* method of the **`UserDialog`** to load the dialog template from the resource script file. The programmer then does not need to worry about the details of how and when to load the resource script. There is a slight difference in *new* the class. The programmer does not need to use the *defineDialog* method because the dialog is defined in the resource script. Other than that, the **`RcDialog`** class is used in the same manner as the **`UserDialog`** class.

Although the programmer does not need to use the *define*() method, there is nothing preventing its use. The *define*() method could be used at runtime to add additional dialog controls to those defined in the resource script.

7.2.1. new (Class method)

```
>>--init(--script--,--id--+-----------+--+---------+--+--------+--+-----------+--)--><
                          +-,-dlgData.-+  +-,-hFile-+  +-,-opts-+  +-,-expected-+
```

Instantiates a **`RcDialog`** object.

Arguments:

The arguments when creating a new dialog instance of this class are:

script [required]
: The file name of the resource script containing the dialog template.

id [required]
: The resource ID of the dialog template. This may be numeric or *symbolic*.

dlgData. [optional]
: A *dialog data stem* variable (don't forget the trailing period) that contains data used to initialize the underlying dialog's controls.

hFile [optional]
: The name of a *file*) containing symbolic ID defines for resource IDs.

opts [optional]
: Zero or more of the following keywords, separated by blanks:

CENTER
: The dialog is to be positioned in the center of the screen.

CONNECTBUTTONS
: Each push *Button* in the underlying dialog has the CLICKED *event* notification connected automatically to a method in the Rexx dialog object. This is the same as using the *connectButtonEvent*() method for the CLICKED notification. The name for the method is generated automatically by the ooDialog framework. The method name is the button label with all spaces, ampersands, colons, and trailing ... characters removed.

CONNECTRADIOS
: Similar to CONNECTBUTTONS, this option connects the CLICKED event notification from each *RadioButton* button to a method in the Rexx dialog object. Again, this is the same as using the *connectButtonEvent* method. For radio buttons, the generated method name is the button label with all spaces, ampersands, colons, and trailing ... characters removed. The resulting text is then **prepend** with the text **`ID`**.

CONNECTCHECKS
: Exactly the same as CONNECTRADIOS, for check *CheckBox* controls. The object method name is generated in the same way as it is for radio buttons. That is, the method name is the button label, with all spaces, ampersands, colons, and trailing ... characters removed. Which is then **prepended** with the text **`ID`**.

expected [optional]

This is the maximum number of dialog controls expected in the dialog template. It serves the same purpose as the *expected* argument in the *create*() method of the **`UserDialog`**. The default value for this argument is 200.

Remarks:

Normally a programmer does not instantiate a **`RcDialog`** directly, but rather creates a subclass of a **`RcDialog`** and instantiates the subclass. Recall that in ooRexx, when a new object is instantiated, the *new* method invokes the *init*() method of the object using the arguments passed to the *new*() method. So, the arguments of the *new* method are also the arguments of the *init* method.

If the programmer over-rides the *init* method in the subclass of the **`RcDialog`**, care must be taken to invoke the superclass *init* method with the correct arguments.

Example:

This example creates a new dialog object using the **`SimpleLB`** class, which subclasses the **`RcDialog`**. It uses the dialog definition with symbolic ID **`IDD_DIALOG1`** in the **`listBox.rc`** resource script file. The dialog is initialized so that the "Doctor" item is selected in the list box with symbolic ID of **`IDC_LB`** by using the **`dlgData.`** stem variable. The symbolic IDs for this program are contained in the "resource.h" file. When the user closes the dialog, the **`dlgData.IDC_LB`** variable will contain the text of the selected item in the list box.

```
/* Simple ListBox using a .rc file for the dialog template */

  dlgData.IDC_LB = "Doctor"

  dlg = .SimpleLB~new("listBox.rc", IDD_DIALOG1, dlgData., "resource.h")
  if dlg~initCode = 0 then do
    dlg~execute("SHOWTOP")
  end
  else do
    say "Problem creating the dialog.  init code:" dlg~initCode
    return 99
  end

  say "The user's chosen profession is" dlgData.IDC_LB
  return 0

::requires "ooDialog.cls"

::class 'SimpleLB' subclass RcDialog

::method initDialog

  lb = self~newListBox(IDC_LB)

  lb~add("Business Manager" )
  lb~add("Software Developer")
  lb~add("Accountant")
  lb~add("Lawyer")
  lb~add("Doctor")
```

To try the example, cut and paste the above code into a file named **`listBox.rex`** and paste the following code into the files **`listBox.rc`** and **`resource.h`**.

```
/* listBox.rc */
#include <windows.h>
#include <commctrl.h>
#include "resource.h"
```

```
IDD_DIALOG1 DIALOGEX 30,30,179,182
STYLE DS_MODALFRAME | DS_FIXEDSYS | WS_VISIBLE | WS_CAPTION | WS_POPUP | WS_SYSMENU
CAPTION "List Box Example"
FONT 8,"MS Shell Dlg 2",400,0,1
BEGIN
    DEFPUSHBUTTON   "Close",IDOK,122,161,50,14
    LISTBOX         IDC_LB,8,28,163,122,WS_TABSTOP | LBS_NOINTEGRALHEIGHT | LBS_SORT
    CTEXT           "Pick Your Chosen Profession",IDC_STATIC,26,13,122,9
END

/* resource.h */
#ifndef IDC_STATIC
#define IDC_STATIC (-1)
#endif

#define IDD_DIALOG1                     100
#define IDC_LB                          1001
```

UserDialog Class

The **`UserDialog`** class is a concrete subclass of the *dialog* object. This class allows the programmer to dynamically define the dialog *template* in memory. The class provides methods to begin the dialog template, to create the dialog controls within the dialog dialog template, and to read and parse a resource *script*. These methods allow the programmer to create a dialog with any of the styles or behaviors supported by the Windows operating system, and to create any of the dialog controls supported by ooDialog.

Note: The class has all the methods of the *dialog* object. Only the methods specific to the **`UserDialog`** are documented here.

The basic process for creating a **`UserDialog`** is this: Invoke the *create* or *createCenter* methods to start the dialog template. Then, in the *defineDialog* method, create each dialog control using one of the *create...* methods. The *defineDialog* method is invoked automatically by the *create* or *createCenter* methods and is meant to be over-ridden by the programmer for the purpose of creating dialog controls.

In addition to creating dialog controls one at a time, there are also methods that allow the programmer to define a group of the same dialog controls together. The names of these methods begin with *create* and usually end with *Group*, *Input*, or *Stem*. There are also methods for creating the Ok and Cancel buttons as a group. These methods begin with *createOkCancel*.

It is also possible to combine reading a dialog definition from a resource script and creating controls dynamically. The easiest way to do this would be to start with a *RcDialog*. This would create the dialog template and some of the dialog controls using a resource script. Since a **`RcDialog`** is a subclass of the **`UserDialog`**, it also has a *defineDialog*() method that is invoked automatically.

The programmer would over-ride the *defineDialog* method and create additional dialog controls using the *create ...* methods. This could be useful, for instance, in an application that had several dialogs with similar features. The similar aspects of each dialog could be placed in a resource script and then the individual dialogs could be customized using the *create ...* methods.

8.1. Method Table

Instances of the **`UserDialog`** class implement the methods listed in the following table.

Table 8.1. UserDialog Method Reference

Dialog Method	Description
Class Methods	**Class Methods**
new	Instantiates a new **`UserDialog`** object.
Instance Methods	**Instance Methods**
addIconResource	Adds the file name for an icon resource to the dialog's icon table.
create	Begins the creation of the in-memory Windows dialog template by specifying the general details for the dialog itself.
createBitmapButton	Creates a bitmap push button in the dialog template.
createBlackFrame	Creates a static black frame control in the dialog template.
createBlackRect	Creates a static black rectangle control in the dialog template.
createCenter	Begins the creation of an in-memory dialog template for a dialog that will be centered in the user's screen.
createCheckBox	Creates a check box control in the dialog template.

Dialog Method	Description
createCheckBoxGroup	Creates a group of check box controls in the dialog template using a single string to specify the check box labels.
createCheckBoxStem	Creates a group of check box controls in the dialog template using a stem to specify the check box labels.
createComboBox	Creates a combobox control in the dialog template.
createComboBoxInput	Creates a combobox with a static text control (for the label) in the dialog template.
createDateTimePicker	Creates a date time picker control in the dialog template.
createEdit	Creates an edit control in the dialog template
createEditInput	Creates an edit control and a static text control (for the label) in the dialog template, in one step.
createEditInputGroup	Creates one or more edit controls with labels in the dialog template using a single string to specify the labels.
createEditInputStem	Creates one or more edit controls with labels in the dialog template using a stem to specify the labels.
createEtchedFrame	Creates a static etched frame control in the dialog template.
createEtchedHorizontal	Creates a static control in the dialog template that is an etched horizontal line.
createEtchedVertical	Creates a static frame control in the dialog template that is a single etched vertical line.
createGrayFrame	Creates a static gray frame control in the dialog template.
createGrayRect	Creates a static gray rectangle control in the dialog template.
createGroupbox	Creates a group box in the dialog template.
createListBox	Creates a list box control in the dialog template.
createListView	Creates a list-view control in the dialog template.
createMonthCalendar	Creates a month calendar control in the dialog template.
createOkCancelRightBottom	Creates an Ok and Cancel push buttons in the right bottom corner of the dialog template.
createOkCancelRightTop	Creates an Ok and Cancel push buttons in the right top corner of the dialog template.
createOkCancelLeftBottom	Creates an Ok and Cancel push buttons in the left bottom corner of the dialog template.
createOkCancelLeftTop	Creates an Ok and Cancel push buttons in the left top corner of the dialog template.
createPasswordEdit	Creates an edit control with the PASSWORD style set in the dialog template.
createProgressBar	Creates a progress bar control in the dialog template.
createPushButton	Creates a push button in the dialog template.
createPushButtonGroup	Creates any number of push buttons in the dialog template at one time.
createPushButtonStem	Creates any number of push buttons in the dialog template at one time, using a **`Stem`** to specify the details for each button.
createRadioButton	Creates a radio button control in the dialog template.

Dialog Method	Description
createRadioButtonGroup	Creates a group of radio buttons with labels in the dialog template using a single string to specify the labels.
createRadioButtonStem	Creates a group of radio buttons with labels in the dialog template using a stem to specify the labels.
createReBar	Creates a rebar control in the dialog template.
createScrollBar	Creates a scroll bar control in the dialog template.
createStatic	Creates a static control in the dialog template.
createStaticFrame	Creates one of the static frame controls in the dialog template.
createStaticImage	Creates a static image control in the dialog template.
createStaticText	Creates a static text control in the dialog template.
createStatusBar	Creates a status bare control in the dialog template.
createTab	Creates a tab control in the dialog template.
createToolBar	Creates a toolbar control in the dialog template.
createTrackbar	Creates a trackbar control in the dialog template.
createTreeView	Creates a tree view control in the dialog template
createUpDown	Creates an up-down control in the dialog template.
createWhiteFrame	Creates a white static frame control in the dialog template.
createWhiteRect	Creates a white static rectangle control in the dialog template.
defineDialog	The programmer uses this method to dynamically create the dialog controls in the dialog template by using the appropriate create ... methods.
load	Creates the in-memory dialog template by reading the dialog template from a resource script file.
loadFrame	Starts the in-memory Windows dialog template by reading a dialog template from a resource script file.
loadItems	Creates the dialog controls in the dialog template, and finishes the dialog template, by reading a dialog template from a resource script file.

8.2. new (Class method)

```
>>--new(--+----------+--+--------------+--)-----><
          +-dlgData.-+  +-,--headerFile-+
```

Instantiates a new **`UserDialog`** object.

Arguments:

The arguments are:

dlgData. [optional]

A *dialog data stem* variable (don't forget the trailing period) that contains data used to initialize the underlying dialog's controls.

hFile [optional]

The name of a *file* containing symbolic ID defines for resource IDs.

Remarks:

Normally a programmer does not instantiate a **`UserDialog`** directly, but rather creates a subclass of a **`UserDialog`** and instantiates the subclass. Recall that in ooRexx, when a new object is instantiated, the *new* method invokes the *init*() method of the object, using the arguments of the *new*() method. Therefore, the arguments of the *new* method are also the arguments of the *init* method.

If the programmer over-rides the *init* method in the subclass of the **`UserDialog`**, care must be taken to invoke the superclass *init* method with the correct arguments.

Example:

The following example creates a new dialog object, adding any symbolic IDs defined in **`resources.h`** to the object's *constDir* directory:

```
  dlg =.PhoneBookDlg~new(aStem., "resources.h")
  ...

::requires "ooDialog.cls"

::class "PhoneBookDlg" subclass UserDialog

...
```

The *example* for the *new* method in the dialog object shows the use of the dialog data stem and header file arguments in more detail.

8.3. addIconResource

```
>>--addIconResource(--id--,--fileName--)---------><
```

The *addIconResource* adds an icon resource file name to the dialog. The icon file name is added by the programmer by directly invoking this method. In addition, when a resource script is parsed, this method is automatically invoked when an ICON resource statement encountered.

Arguments:

The arguments are:

id [required]
: The *resource ID* for the icon. This can be numeric or *symbolic*.

fileName [required]
: The file name of the icon. If the icon file is not in the current directory, then a relative or fully qualified file name should be used.

Return value:

The possible return values are:

0
: Success.

-1
: A problem with the resource ID. Either a symbolic ID could not be resolved, or the ID is reserved.

Remarks:

Once the icon file name is added to the dialog it will still need to be loaded into memory to be of use in a Rexx program. There are two ways in which this can be done:

1. The icon can be used as the *dialog icon*. Specify the resource ID of the icon as the *icon* argument in the *execute*, *executeAsync*, *popup*, or *popupAsChild* methods. The ooDialog framework will automatically load the icon.

2. The icon can be loaded into memory by using the *userIcon* method of the *Image* class.

Example:

The following example creates a very simple dialog that has an OK button and a static text box. The *addIconResource* method is used to load an icon resource using the ID of 105. That icon is then used in the *execute* method for the *dialog icon*.

```
  dlg = .SimpleDialog~new
  if dlg~initCode = 0 then do
    dlg~addIconResource(105, "MyAppIcon.ico")
    dlg~create(30, 30, 150, 100, "The Simple Dialog", "VISIBLE")
    dlg~execute("SHOWTOP", 105)
  end

::requires "ooDialog.cls"

::class 'SimpleDialog' subclass UserDialog

::method defineDialog

  self~createStaticText(100, 10, 20, 130, 30, "CENTER BORDER", "Simple message")
  self~createPushButton(IDOK, 105, 70, 35, 15, "DEFAULT", "OK")
```

8.4. create

```
>>--create(--x--,--y--,--cx--,--cy--+---------+--+---------+--+------------+--->
                                    +-,-title-+  +-,-style-+  +-,-dlgClass-+

>----+------------+--+------------+--+------------+--)-------------------------><
     +-,-fontName-+  +-,-fontSize-+  +-,-expected-+
```

The *create* method begins the creation of the in-memory Windows dialog *template* by specifying the general details for the dialog itself.

Once the dialog template has been started, the dialog controls can be created in the dialog template using the *create...* methods.

Arguments:

The arguments are:

x, y [required]
: The position of the upper-left corner of the dialog in dialog units.

cx, cy [required]
: The width and height of the dialog in dialog units.

title [optional]
: The title for the dialog. If omitted, the dialog will not have a title.

style [optional]

Zero or more of the keywords listed below, separated by blanks:

VISIBLE

Creates the dialog in a visible state. The default is to create the dialog invisible. The operating system does not show the dialog until the visible state is set, and the programmer sets the dialog to visible when desired.

A common technique in Windows is to create the dialog invisible, initialize all the dialog controls, and then make the dialog visible. To use this technique in ooDialog, the programmer would not use the VISIBLE keyword in *create* or *createCenter*. Then in the method that runs the dialog, *execute*, *popup*, *popupAsChild*, etc., make the dialog visible through the use of the *show* argument. In those methods, the ooDialog framework invokes *initDialog* before it shows the dialog. During the execution of *initDialog* the dialog is invisible on the screen.

On the other hand, if the programmer **does** want the dialog visible during *initDialog*, the VISIBLE keyword should be used.

NOMENU

Creates a dialog without a system menu. By default the dialog is created with a system menu.

NOTMODAL

Creates a dialog with a normal dialog frame. This option is not needed, and is probably a hold over from the original ooDialog that ran on Windows 3.1. The programmer will not see any difference in the dialog frame whether this option is used or not used.

SYSTEMMODAL

In many ways this option is also obsolete. It adds a dialog style that was used in Windows 3.1. In Windows 3.1, the style would create a dialog that blocked access to all other windows. In modern versions of Windows this is no longer the case, with this style the user can still access other windows. However, with this style, the dialog will remain on top of all other windows in the system, even when the user is working with another window. This is an unusual state that may frustrate the user.

CENTER

Creates a dialog that is centered in the user's screen. When this keyword is used, the operating system ignores the x, y position arguments. The *createCenter* convenience method can also be used to create a centered dialog.

THICKFRAME

Creates a dialog with a thick frame that can be used to resize the dialog.

MINIMIZEBOX

Creates a dialog with a minimize button.

MAXIMIZEBOX

Creates a dialog with a maximize button.

VSCROLL

Creates a dialog that has a vertical scroll bar.

HSCROLL

Creates a dialog with a horizontal scroll bar.

OVERLAPPED

This keyword is accepted, but like the NOTMODAL keyword, it has no discernable effect.

dlgClass [optional]

Name of the window class used for the dialog. This argument is, and always has been, ignored. It is difficult to say what the original intent for the argument was.

fontName [optional]

The name of the font to be used by the dialog for all text. If this argument is omitted, the default dialog font is used. The programmer can change this default by using *setDefaultFont* class method. If the default font has not been changed, ooDialog uses a symbolic font name provided by Microsoft. This symbolic name signals the operating system to use the appropriate dialog font for the specific version of Windows in use. I.e., the dialog font under Windows 2000 might be different then the font under Windows XP.

fontSize [optional]

The point size of the font used by the dialog. When this argument is omitted, the default dialog font size is used, in the same manner as the font name.

expected [optional]

This argument specifies the maximum number of dialog controls the dialog template is expected to contain. Since the dialog template is constructed in memory, ooDialog needs to have some idea of the size of the system resources to reserve for the template. The default value is 200.

This value is just a means of doing a rough estimate of the memory needed for the template. The amount of memory for each dialog control varies depending on how much, if any, text is associated with the control. If your dialog has more than 200 controls, you should probably set the expected value larger.

As the dialog template is being constructed, the ooDialog framework will detect if the template is about to extend past its allocated storage. If so, an error condition is raised. The condition text will look similar to:

```
Error 98 running C:\Program Files\ooDialog.cls line xxxx:  Execution error
Error 98.900:  the storage allocated for the dialog template is too small
```

If this condition is raised, the programmer needs to use a larger value for the *expected* argument.

Return value:

This method returns true on success, otherwise false. However, if the method fails a condition is raised.

Example:

The following example creates a dialog with a size of 300 by 200 dialog units. It is positioned very near the top and the left of the screen. The dialog has no system menu, minimize box, or maximize box. The dialog will use a 12-point Courier font and is expected to contain fewer than 100 dialog controls.

```
success = dlg~create(20, 20, 300, 200, "My first Dialog", "NOMENU", , "Courier", 12, 100)
if success then dlg~execute
```

8.5. createCenter

```
>>--createCenter(--cx--,--cy--,--+---------+--+---------+--+------------+------>
                                 +-,-title-+  +-,-style-+  +-,-dlgClass-+

>----+------------+--+------------+--+------------+--)-------------------------><
     +-,-fontName-+  +-,-fontSize-+  +-,-expected-+
```

The *createCenter* method begins the creation of an in-memory dialog *template* for a dialog that will be centered in the user's screen. It is a convenience method that merely invokes the *create* method with the CENTER keyword added to the options argument. The *x* and *y* arguments are not needed because they are ignored when the dialog is centered. Other than that the method is exactly the same as the *create* method.

Arguments:

The arguments are:

cx, cy [required]

The width and height of the dialog in dialog units.

title [optional]

The title for the dialog. If omitted, the dialog will not have a title.

style [optional]

Zero or more of the keywords listed below, separated by blanks:

VISIBLE

Creates the dialog in a visible state. The default is to create the dialog invisible. The operating system does not show the dialog until the visible state is set, and the programmer sets the dialog to visible when desired.

A common technique in Windows is to create the dialog invisible, initialize all the dialog controls, and then make the dialog visible. To use this technique in ooDialog, the programmer would not use the VISIBLE keyword in *create* or *createCenter*. Then in the method that runs the dialog, *execute*, *popup*, *popupAsChild*, etc., make the dialog visible through the use of the *show* argument. In those methods, the ooDialog framework invokes *initDialog* before it shows the dialog. During the execution of *initDialog* the dialog is invisible on the screen.

On the other hand, if the programmer **does** want the dialog visible during *initDialog*, the VISIBLE keyword should be used.

NOMENU

Creates a dialog without a system menu. By default the dialog is created with a system menu.

NOTMODAL

Creates a dialog with a normal dialog frame. This option is not needed, and is probably a hold over from the original ooDialog that ran on Windows 3.1. The programmer will not see any difference in the dialog frame whether this option is used or not used.

SYSTEMMODAL

In many ways this option is also obsolete. It adds a dialog style that was used in Windows 3.1. In Windows 3.1, the style would create a dialog that blocked access to all other windows. In modern versions of Windows this is no longer the case, with this style the user can still access other windows. However, with this style, the dialog will remain on top of all other windows in the system, even when the user is working with another window. This is an unusual state that may frustrate the user.

THICKFRAME
Creates a dialog with a thick frame that can be used to resize the dialog.

MINIMIZEBOX
Creates a dialog with a minimize button.

MAXIMIZEBOX
Creates a dialog with a maximize button.

VSCROLL
Creates a dialog that has a vertical scroll bar.

HSCROLL
Creates a dialog with a horizontal scroll bar.

OVERLAPPED
This keyword is accepted, but like the NOTMODAL keyword, it has no discernable effect.

dlgClass [optional]
Name of the window class used for the dialog. This argument is, and always has been, ignored. It is difficult to say what the original intent for the argument was.

fontName [optional]
The name of the font to be used by the dialog for all text. If this argument is omitted, the default dialog font is used. The programmer can change this default by using *setDefaultFont* class method. If the default font has not been changed, ooDialog uses a symbolic font name provided by Microsoft. This symbolic name signals the operating system to use the appropriate dialog font for the specific version of Windows in use. I.e., the dialog font under Windows 2000 might be different then the font under Windows XP.

fontSize [optional]
The point size of the font used by the dialog. When this argument is omitted, the default dialog font size is used, in the same manner as the font name.

expected [optional]
This argument specifies the maximum number of dialog controls the dialog template is expected to contain. Since the dialog template is constructed in memory, ooDialog needs to have some idea of the size of the system resources to reserve for the template. The default value is 200.

This value is just a means of doing a rough estimate of the memory needed for the template. The amount of memory for each dialog control varies depending on how much, if any, text is associated with the control. If your dialog has more than 200 controls, you should probably set the expected value larger.

As the dialog template is being constructed, the ooDialog framework will detect if the template is about to extend past its allocated storage. If so, an error condition is raised. The condition text will look similar to:

```
Error 98 running C:\Program Files\ooDialog.cls line xxxx:  Execution error
Error 98.900:  the storage allocated for the dialog template is too small
```

If this condition is raised, the programmer needs to use a larger value for the *expected* argument.

Return value:

This method returns true on success, otherwise false. However, if the method fails a condition is raised.

Example:

The following example creates a dialog with a size of 300 by 200 dialog units and centered on the screen. The dialog has no system menu, minimize box, or maximize box. The dialog will use a 12-point Courier font and is expected to contain fewer than 100 dialog controls.

```
success = dlg~create(20, 20, 300, 200, "My first Dialog", "NOMENU", , "Courier", 12, 100)
if success then dlg~execute
```

8.6. Create... Methods

8.6.1. Understanding the Create Methods

The methods listed in this section (all starting with create) are used to help create a dialog *template* dynamically. The methods in this section are used to define the individual dialog controls. The methods can also be used in a dialog that is created by loading a resource *script* in a *RcDialog*. The methods will then be used to define controls in addition to the ones defined in the resource script.

During the *effort* to unify the method names in ooDialog, all the add... methods were renamed to create... methods. Where needed, some of these methods had their argument order changed so that the order was consistent for all the create... methods. The create... methods all follow these general guidelines

- The names of the methods all have the format **`create[ControlName]`** for a single control or **`create[ControlName][GroupIndicator]`** for a group of controls where **`[ControlName]`** is the name of the individual control and **`[GroupIndicator]`** hints at what the group of controls is and how the controls are passed to the method.
- For all the create... methods, the arguments are always in the same order. Although, for some of the methods, some arguments may be required and in other of the methods the arguments may be optional. The order is exactly: resourceID, control *coordinates*, style option, text if any, attribute name if any, methodName to connect, if any.

The recommended way to create a dialog dynamically is to subclass the **`UserDialog`** class and put all create... statements into the *defineDialog* method. The ooDialog framework invokes *defineDialog* automatically when the dialog is about to be created.

When *automatic* data field is on, the create... methods automatically create and connect the associated data *attribute* by invoking the matching *connnect data attriubte* methods.

The create... methods cannot be invoked until after the dialog template is started. This is after the *create* or *createCenter* methods have completed successfully. In addition, the methods cannot be used once the underlying dialog has been created. In general this is after the *execute* or the *executeAsync* methods have been invoked. If the programmer puts all the create... methods in the *defineDialog* method as recommended, then he does not need to worry about invoking the methods at the wrong time.

Style argument:

The behavior and appearance of all dialog controls defined using one of the create... methods are controlled by a style option. For each of the create... methods, one of the arguments consists of a set of keywords that define the style of the control being defined. These keywords allow the programmer to change the default behavior and / or appearance of the control.

An attempt is made for each method to list all the style keywords that will be accepted. However, the behaviour and appearance that the specified style gives any particular control is determined by the operating system, not ooDialog. For many of the dialog controls, certain combinations of styles do not make much sense. It is up to the individual programmer to decide whether a style is appropriate for her program.

There are five styles that all dialog controls share:

- Visible
- Enabled
- Border
- Group
- TabStop

The following style keyword list explains these styles and how the keywords are used for the styles. In many cases, there is a common default value for the style and the programmer only needs to specify a keyword when she wants a non-default value for the style. For instance, normally all dialog controls are created enabled. The programmer would only need to specify the DISABLED keyword when the control should be disabled as the dialog is first displayed.

HIDDEN

All windows (and every dialog control is a window) can be visible or invisible. When a control is invisible it is not displayed on the screen and the user will not see it. The control is also not active and will not accept any keyboard or mouse input. But, the control still exists and does not lose any of its internal state. Since the control is invisible, whatever is underneath it will be displayed on the screen. Normally this would be the owner dialog. However, it could also be another dialog control. If this other dialog control is visible, the user will see and can interact with it rather than the invisible control.

In ooDialog by default all controls created using the create... methods will be created visible. To change the default for any control when using the create... methods add the HIDDEN keyword to the style argument. This will make the control invisible when the dialog is first shown on the screen.

For any control, the visible style can be changed during program execution by using the *hide* or *show* methods. (Or any of the related methods like *hideFast*, *display*, etc..)

DISABLED

All windows can be disabled or enabled. When a dialog control is disabled it will not accept keyboard or mouse input and therefore the user can not interact with it. The operating system will draw disabled controls in a different manner than enabled controls to give the user a visual clue that the control is disabled.

When using the create... methods controls will be created enabled. To change this for any control add the DISABLED keyword to the style argument. The control will then be disabled when the dialog is first shown on the screen.

All controls can have their enabled style changed during program execution by using the *disable* or *enable* methods. (Or any of the related methods like *disableControl*, *enableControl*, etc..)

BORDER, NOBORDER

Most, but not all, windows are drawn with a border. When using the create... methods individual controls are drawn by default with, or without, a border. The default is changed by using either the BORDER or NOBORDER keywords. The reference for each individual create... method notes the default for the control for this style and which keyword is used to change the default. In general, this style can not be changed after a control is created.

GROUP

This style is used to place a series of consecutive dialog controls in a group. The first control with the group style starts a new group of controls. This group continues for each control that does not have the group style. As soon as a successive control is encountered with the group style, a new group is started. This style is only meaningful for dialog control windows.

The Windows dialog manager uses this style to determine the behavior of a dialog, mostly the navigation behavior. Within a group of controls the user navigates from one control to the other using the arrow keys rather than the tab key.

The GROUP keyword is used to give a control the group style when it is created using one of the create... methods. After the control has been created its group style can be changed during the execution of a dialog by using the *setGroup* method of the PlainBaseDialog class or the *group* method of the DialogControl class.

TAB, NOTAB

The tabstop style is another style that is only meaningful with dialog controls. When a control has the tabstop style the user can navigate to it using the tab key. If a control does not have this style, the tab key will skip over the control. Dialog controls in the same class usually all have the same style for tabstop. As an example, all button controls usually have the tabstop style, whereas all static text controls usually do not have the tabstop style.

When a control is added to a **`UserDialog`** instance it is given the tabstop style that is normal for the control. The individual create... methods document the default for the control. The programmer uses either the TAB or NOTAB keywords to change the default. After controls have been created they can have their tabstop style changed using the *setTabStop* method of the *dialog* object or the *tabStop* method of the dialog *control* object.

All dialog controls have a position and size.

Coordinates

The position of the control is specified by the coordinates of its upper left corner (x, y) and the size is specified by its width (cx) and height (cy). Windows uses a coordinate system that specifies the upper left corner of the screen as (0,0). Moving to the left in this system increases the x value and moving down increases the y value. Negative values for either x or y would then move off the screen. (Normally. This simple explanation becomes more complicated with dual-monitor systems.)

For the create... methods all coordinates, (the x, y, cx, and cy arguments,) are specified in *dialog unit*s. The position of the dialog control is specified in relation to the upper left corner of the dialog's *client area*. All the create... methods that create a single control have the following common arguments. Some of the other create... methods that create a group of controls will only take the position, or the size arguments:

x

The left position of the control in the dialog's client window, in *dialog unit*s.

y

The top position of the control in the dialog's client window, in dialog units.

cx

The width of the control in dialog units.

cy

The height of the control in dialog units.

id

All dialog controls have a numeric *resource ID*. The ooDialog framework is designed to accept a *symbolic* ID anywhere a resource ID is needed. All the create... methods accepted a symbolic ID for the id argument.

By default static controls and group boxes are assigned a resource id of -1. However, assigning a positive id allows the programmer to modify the static control or group box after it is created, perhaps to change the text of its title, or to change its position. Without a positive ID, a static control or group box can not be modified. Which is fine, normally static controls and group boxes do not need to be modified.

8.6.2. Common Control Styles

Windows has a set of common control styles. ooDialog has support for using these styles when creating the *ReBar*, *StatusBar*, and *ToolBar* dialog controls in a **`UserDialog`**. The Microsoft *documentation* says except where noted, these styles apply to header controls, toolbar controls, and status windows. However, some of the styles seem to also effect the rebar control.

In addition, the Microsoft documentation is rather vague as to exactly what styles effect each control. Some styles seem to have no effect, other styles obviously do have an effect. The ooDialog framework parses the *style* argument for all the following keywords and adds that style when creating the dialog control. In many cases the ooDialog programmer will have to experiment to determine for herself what effect, if any, the style has on a control.

The following keywords, separated by blank, will add the common control style to a rebar, status bar, or tool bar control. Case is not significant:

CCS_ADJUSTABLE	CCS_NOMOVEX	CCS_RIGHT
CCS_BOTTOM	CCS_NOMOVEY	CCS_TOP
CCS_LEFT	CCS_NOPARENTALIGN	CCS_VERT
CCS_NODIVIDER	CCS_NORESIZE	

CCS_ADJUSTABLE

Enables a toolbar's built-in customization features, which allow the user to drag a button to a new position or to remove a button by dragging it off the toolbar. In addition, the user can double-click the toolbar to display the Customize Toolbar dialog box, which enables the user to add, delete, and rearrange toolbar buttons.

CCS_BOTTOM

Causes the control to position itself at the bottom of the parent window's client area and sets the width to be the same as the parent window's width. Status bar windows have this style by default.

CCS_LEFT

Causes the control to be displayed vertically on the left side of the parent window.

CCS_NODIVIDER

Prevents a two-pixel highlight from being drawn at the top of the control.

CCS_NOMOVEX
: Causes the control to resize and move itself vertically, but not horizontally, in response to a WM_SIZE message. If CCS_NORESIZE is used, this style does not apply.

CCS_NOMOVEY
: Causes the control to resize and move itself horizontally, but not vertically, in response to a WM_SIZE message. If CCS_NORESIZE is used, this style does not apply. Header windows have this style by default.

CCS_NOPARENTALIGN
: Prevents the control from automatically moving to the top or bottom of the parent window. Instead, the control keeps its position within the parent window despite changes to the size of the parent. If CCS_TOP or CCS_BOTTOM is also used, the height is adjusted to the default, but the position and width remain unchanged.

CCS_NORESIZE
: Prevents the control from using the default width and height when setting its initial size or a new size. Instead, the control uses the width and height specified in the request for creation or sizing.

CCS_RIGHT
: Causes the control to be displayed vertically on the right side of the parent window.

CCS_TOP
: Causes the control to position itself at the top of the parent window's client area and sets the width to be the same as the parent window's width. Toolbars have this style by default.

CCS_VERT
: Causes the control to be displayed vertically.

8.6.3. Create Button Controls

The methods in this section are all used to create button controls in the dialog *template*. *Button* controls include push buttons, radio buttons, check boxes, group boxes, and owner-drawn buttons.

8.6.3.1. createBitmapButton

```
>>--createBitmapButton(-id-,-x-,-y-+------+-+------+-+----------+-+--------+--->
                                   +-,-cx-+ +-,-cy-+ +-,-style--+ +-,-text-+

>--+------------+-,-normal-+------------+--+-------------+--+-------------+--)--><
   +-,-methName-+          +-,-focused--+  +-,-selected--+  +-,-disabled--+
```

The *createBitmapButton* method creates a bitmap push button in the dialog template. A bitmap push button uses a bitmap instead of plain text for the face of the button. You can provide four different bitmaps representing the four states of a button. The bitmap push button is an owner-drawn button, where ooDialog internally handles the drawing of the button.

Note: On Windows XP and later, using an *image* list with the button will produce a better overall appearance in the dialog. Using an image list gives the button the same look and feel as other buttons on the system. The buttons produced using the *createBitmapButton* and the *installBitmapbutton* methods have the look and feel of Windows 98 buttons. When an image list is used with a button, the other button styles, such as the NOTIFY style can still be used. Whereas with the bitmap button the other button styles can not be used. The image list button supports all the features of the bitmap button, and more.

The bitmap arguments can be specified by either a file name, a bitmap handle, or the numeric resource ID of a bitmap in the resource DLL of a *ResDialog*. You can retrieve a bitmap handle by loading a bitmap stored in a file into memory, using the *loadBitmap* method. If you pass a bitmap handle to the method, you must use the INMEMORY option. If you use a resource ID for the bitmaps, the underlying dialog must have already been created.

Arguments:

The arguments are:

id [required]

The *resource ID* for the control.

x, y [required]

The control's position *coordinates*.

cx, cy [optional]

The control's width and height *coordinates*. If omitted the button will be created with 0 width and height. The programmer would be responsible for resizing the button after the underlying dialog is created. This makes sense when the programmer wants to size the button to match the exact size of the loaded bitmap(s).

style [optional]

Note: The 3.2.0 ooDialog documentation incorrectly stated you could use the NOTIFY style keyword. Do not do this. Owner-drawn buttons should not use any other button styles than OWNER or DEFAULT, and the OWNER style is automatically added by the ooDialog framework. The result of using the NOTIFY style will be less than satisfactory. A button that uses an image *list* allows the use of the NOTIFY style and will give a better overall look to the dialog. Using a button with an image list should be the preferred method for non-text buttons.

The *style* argument can be 0 or more of the following keywords:

DEFAULT	STRETCH	GROUP
FRAME	HIDDEN	TAP
USEPAL	DISABLED	NOTAB
INMEMORY	BORDER	

DEFAULT

The button becomes the default button in the dialog.

FRAME

The button is drawn with a 3D frame. Internally, ooDialog draws the button. However, the technique used is now out of date. This gives your bitmap the same appearance as buttons in Windows 98. To have your buttons match the Window XP and Vista styles use an *image* list with the button as described in the opening comments above.

USEPAL

The color palette of the bitmap is loaded and used. This argument should be specified for just one of the dialog buttons, because only one color palette can be active at any time.

INMEMORY

Specifies that the bitmap is already loaded into memory. If you switch often between different bitmaps within one button, the loading of all bitmaps into memory increases performance.

STRETCH
: If this option is specified and the extent of the bitmap is smaller than the extent of the button rectangle, the bitmap is adapted to match the extent of the button.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

BORDER
: The *border* window style.

GROUP
: The *group* control style.

NOTAB
: The no *tabstop* control style. By default the button is created with the tabstop style except in these conditions: the bmpFocused is omitted, and the bmpSelected is omitted, and the FRAME style keyword is not used.

TAB
: The *tabstop* control style. The button is normally created with the tabstop style and this keyword is not necessary. However, if the bmpFocused argument is omitted, and the bmpSelected argument is omitted, and the FRAME style keyword is not used, the button is created without the tabstop style. In this case only, it is necessary to use the TAB keyword to change the default behavior.

text [optional]
: The text label for the button. Although the user will not see this text, (they see the bitmap instead,) the text is still associated with the button and can be retrieved by the programmer using the *getText* method.

methName [optional]
: The name of a method in the Rexx dialog class that is to be invoked each time the button is clicked. This serves the same purpose as the *connectButtonEvent* method for the CLICKED event. If this argument is omitted, then no method is associated with the CLICKED event.

bmpNormal [required]
: A bitmap that is displayed when the button is in the normal state. I.e., it is not focused, not selected, and not disabled. This argument must be specified. If any of the other three bitmap arguments are omitted, this bitmap will be used in its place.

focused [optional]
: A bitmap that is displayed if the button is focused. When a button is focused it receives the keyboard input and can be clicked by using the Enter or Space keys. Normally the focused button is surrounded by a dashed frame. If this argument is omitted, the **`normal`** bitmap is displayed for the focused state.

selected [optional]
: A bitmap that is displayed while the button is clicked and held down. If this argument is omitted, the **`normal`** bitmap is displayed for the selected state.

disabled [optional]
: A bitmap that is displayed if the button is disabled. If this argument is omitted, the **`normal`** bitmap is displayed for the disabled state.

Return value:

The possible return values are:

0
: Success.

less than 0
: Error with the in-memory dialog template or with the resource ID.

Example:

The following example defines a button with ID 601. The bitmap in the **`Button1.bmp`** file is displayed for the push button in the normal state. If the button is disabled by using the *disableControl* method, the **`Button1D.bmp`** is shown. The *onBmpPushed* method in the Rexx dialog is connected to the CLICKED event. When the button is clicked, the *onBmpPushed* method is invoked.

```
self~createBitmapButton(601, 20, 317, 80, 30, "FRAME USEPAL", , "onBmpPushed", -
                        "Button1.bmp", , , "Button1D.bmp")
```

8.6.3.2. createCheckBox

```
>>--createCheckBox(-id-,--x-,--y--+------+--+------+--+---------+--+---------+->
                                  +-,-cx-+  +-,-cy-+  +-,-style-+  +-,-label-+

>----+-----------------+--+--------------+--)----------------------------------><
     +-,-attributeName-+  +-,-connectMth-+
```

The *createCheckBox* method creates a check box in the dialog template.

By default the check box will have the AUTOMATIC check box style. With this style, the operating system automatically checks or unchecks the control when the user changes the check state. The opposite of the this style is the STANDARD check box. With the standard check box, the programmer is responsible for managing the check state of the control. The 3STATE style, creates the check boxes an automatic three-state check box, also with the AUTOMATIC style. Standard and standard three-state check boxes can not be created through this method. See the *CheckBox* class for more information on check box controls.

Arguments:

The arguments are:

id [required]
: The *resource ID* for the control.

x [required], y [required], cx [optional], cy [optional]
: The control *coordinates* If cx or cy are omitted, then the size for the check box is calculated internally using its label.

style [optional]
: A list of 0 or more of the following *style* keywords separated by spaces:

3STATE	TOP	LTEXT
OWNER	BOTTOM	RBUTTON

BITMAP	VCENTER	HIDDEN
ICON	MULTILINE	DISABLED
LEFT	NOTIFY	BORDER
RIGHT	PUSHLIKE	GROUP
HCENTER	FLAT	NOTAB

3STATE

A three-state check box is created. All check box controls have two states, checked and cleared (not checked.) A three-state check box also has an indeterminate state. This is represented by a grayed box inside the check box. The programmer can use methods of the *CheckBox* class to determine which of the 3 states a check box is in.

OWNER

The programmer is completely responsible for drawing the button when the dialog receives the WM_DRAWITEM message. Currently this would be difficult (but not impossible) to implement in ooDialog. For a check box, the difficulty would probably make this impractical.

ICON

The check box displays an icon image.

BITMAP

The check box displays a bitmap image.

LEFT

Left justifies the text to the right of the check box.

RIGHT

Right justifies the text to the right side of the check box.

HCENTER

The text is centered horizontally to the right of the check box. This is the default if neither LEFT nor RIGHT are specified.

TOP

The text is aligned at the top of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

BOTTOM

The text is aligned at the bottom of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

VCENTER

The text is vertically centered in the button rectangle. This is the default if neither TOP nor BOTTOM are specified.

MULTILINE

If the text for the label of the button is longer than the width of the button rectangle, the operating system will word wrap the text producing more than one line. The height of the button rectangle has to be sufficient to display the extra line(s) or the text is clipped.

NOTIFY

Enables the button to send notifications for the gained focus,lost focus, and double click events. This is only necessary when the *connectButtonEvent* method is used, and only for the GOTFOCUS, LOSTFOCUS, or DBLCLK keywords of that method. Note that only radio buttons and owner-drawn buttons will receive the DBLCLICK event.

PUSHLIKE
: Makes the button look and act like a push button. The button looks raised when it isn't pushed or checked, and sunken when it is pushed or checked. The button is drawn with a flat appearance, i.e., it is drawn as a flat rectangle the label text inside the rectangle.

FLAT
: The button is drawn with a flat appearance, i.e., it is drawn as a flat rectangle the label text inside the rectangle.

LTEXT
: Places text on the left side of the check box. This is the same as the RBUTTON style.

RBUTTON
: Positions the check box's square on the right side of the button rectangle. This is the same as the LTEXT style.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

BORDER
: The *border* window style.

GROUP
: The *group* control style.

NOTAB
: The no *tabstop* control style.

label [required]
: The text that is displayed next to the check box.

attributeName [optional]
: The name of a data *attribute* to connect with the check box. Using this argument allows the programmer to control the name of the data attribute. If you omit this argument, or the supplied name is not valid, then the data attribute name is constructed *automatically*.

  If automatic data *initAutoDetection* is off then no data attribute is created, whether this argument is used or not. The *connectCheckBox* method can be used to manually create and connect the data attribute when data detection is off.

connectMth [optional]
: A text string that allows the *connectButtonEvent* CLICKED event notification to be automatically connected to a method in the dialog. In normal usage *connectMth* is the name of the method in the Rexx dialog to be connected to the CLICKED event.

  However, the *createCheckBox* method is also used by the *RcDialog* to create check box controls found in the resource *script* for the dialog. Therefore, if this string contains the word CONNECTCHECKS, case insignificant, then the method name is generated in the same way as it is by specifying CONNECTCHECKS in the *opts* argument when *new* a **`RcDialog`**.

  First a string is constructed from the label of the button with all space, tab, ampersand, colon, and plus sign characters removed. In addition, if the label were to have a trailing ... that is also removed. Then that string is prepended with the text **`id`**. This string is used as the method name to connect the clicked event to.

Return value:

The possible return values are:

-2

The dialog template has either not been started or is complete.

-1

A symbolic ID was used, but it could not be resolved correctly.

0

Success.

1

The control was added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

Example:

The following example defines an automatic three-state check box. The label for the check box is rather long, so a multiline style is used.

```
::method defineDialog

label = "Use condensed type only"
style = "3STATE MULTILINE NOTIFY"
self~createCheckBox(IDC_CB_CONDENSED, 125, 19, 30, 20, style, label)
```

Note;

There are also two methods that create a whole group of check boxes automatically, the *createCheckBoxGroup*() and *createCheckBoxStem* methods.

8.6.3.3. createCheckBoxGroup

```
                                             +-----+
                                             V     |
>>--createCheckBoxGroup(id,-x-,-y-+------+-,---txt-+--+---------+-+----------+-)--><
                                  +-,-cx-+            +-,-style-+ +-,-idStat-+
```

The *createCheckBoxGroup* method creates a group of check boxes in the dialog template. This method behaves in a similar fashion to the *createRadioButtonGroup*() method.

Arguments:

The arguments are:

id [required]

The *resource ID* for the first check box. This id is increased by 1 for each additional check box and then assigned to the control.

x, y [required]

The position *coordinates* of the group of check boxes. x and y name the position of the first check box control. The other check boxes are positioned automatically.

cx [optional]

The width *coordinates* of as single check box, plus its text. If omitted, the space needed for the check box and its text is calculated by the ooDialog framework.

txt [required]

A single string that specifies the text for the label of each check box. The string consists of single words separated by blank spaces. This argument determines the number of check boxes in total. Each single word is the label for one check box. Note that this prevents using more than one word for the label of any check box.

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces. Each check box in the group is given the style specified by this option.

3STATE	TOP	LTEXT
OWNER	BOTTOM	RBUTTON
BITMAP	VCENTER	HIDDEN
ICON	MULTILINE	DISABLED
LEFT	NOTIFY	NOBORDER
RIGHT	PUSHLIKE	GROUP
HCENTER	FLAT	NOTAB

3STATE

A three-state check box is created. All check box controls have two states, checked and cleared (not checked.) A three-state check box also has an indeterminate state. This is represented by a grayed box inside the check box. The programmer can use methods of the *CheckBox* class to determine which of the 3 states a check box is in.

OWNER

The programmer is completely responsible for drawing the button when the dialog receives the WM_DRAWITEM message. Currently this would be difficult (but not impossible) to implement in ooDialog. For a check box, the difficulty would probably make this impractical.

ICON

The button displays an icon image.

BITMAP

The button displays a bitmap image.

LEFT

Left justifies the text to the right of the check box.

RIGHT

Right justifies the text to the right side of the check box.

HCENTER

The text is centered horizontally to the right of the check box. This is the default if neither LEFT nor RIGHT are specified.

TOP

The text is aligned at the top of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

BOTTOM
: The text is aligned at the bottom of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

VCENTER
: The text is vertically centered in the button rectangle. This is the default if neither TOP nor BOTTOM are specified.

MULTILINE
: If the text for the label of the button is longer than the width of the button rectangle, the operating system will word wrap the text producing more than one line. The height of the button rectangle has to be sufficient to display the extra line(s) or the text is clipped.

NOTIFY
: Enables the button to send notifications for the gained focus,lost focus, and double click events. This is only necessary when the *connectButtonEvent* method is used, and only for the GOTFOCUS, LOSTFOCUS, or DBLCLK keywords of that method. Note that only radio buttons and owner-drawn buttons will receive the DBLCLICK event.

PUSHLIKE
: Makes the button look and act like a push button. The button looks raised when it isn't pushed or checked, and sunken when it is pushed or checked. The button is drawn with a flat appearance, i.e., it is drawn as a flat rectangle the label text inside the rectangle.

FLAT
: The button is drawn with a flat appearance, i.e., it is drawn as a flat rectangle the label text inside the rectangle.

LTEXT
: Places text on the left side of the check box. This is the same as the RBUTTON style.

RBUTTON
: Positions the check box's square on the right side of the button rectangle. This is the same as the LTEXT style.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

NOBORDER
: Use this keyword to specify that a group *GroupBox* is not added around the check box group.

GROUP
: Do not use this keyword with this method. The group style is added correctly internally.

NOTAB
: The no *tabstop* control style.

idstat [optional]
: This argument is used to set the *resource ID* for the group *GroupBox*, if one is used. The argument is ignored if the NOBORDER style is used. If the NOBORDER style is not used and this argument is omitted then -1 is used for the resource ID.

Example:

The following example adds a group of four check boxes to the dialog. Two check boxes are *preselected*/>:

```
self~createCheckBoxGroup(401, 23, 18, ,"Smalltalk C++ ObjectRexx OO-COBOL")
self~smalltalk = 1
self~objectrexx = 1
```

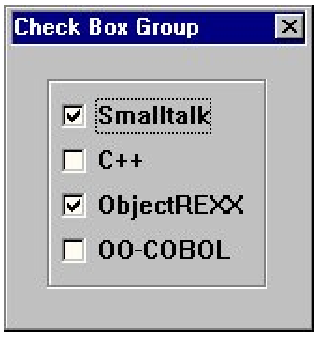


Figure 8.1. Sample Check Box Group

8.6.3.4. createCheckBoxStem

```
>>--createCheckBoxStem(id,-x-,-y-+------+-,-txt.-+-------+-+---------+-+----------+-)-><
                                 +-,-cx-+        +-,-max-+ +-,-style-+ +-,-idStat-+
```

The *createCheckBoxStem* method creates a group of check box controls. Unlike the *createCheckBoxGroup* method, you pass the text for the labels of the check boxes in a stem variable, instead of using a string. This allows you to use labels that include blanks. This method is similar to the *createRadioButtonStem* method, but is used for check boxes.

Note: The documentation prior to version 4.0.0 listed a font name and a font size argument. These arguments do nothing and are ignored in ooDialog 4.0.0 and later.

Arguments:

The arguments are:

id [required]

The *resource ID* for the first check box This id is increased by 1 for each additional check box and then assigned to the control.

x, y [required]

The position *coordinates* for the check box group. The group is positioned so that the upper left corner of the group is at the x, y coordinates specified.

cx [optional]

The width *coordinates* for a single check box plus its label. If omitted, the proper width is calculated by the ooDialog framework.

txt. [required]

A stem containing the text for the label of each check box. The stem indexes should start at 1 and continue consecutively, with each succeeding number containing the label for the next check box. This argument determines the number of check boxes in total.

max [optional]

A number specifying the maximum check boxes to put in a column. The programmer can use this to balance the look of the group by placing the check box in more than 1 column. For instance, if there are 8 check boxes and the max argument is 4 then the result will have 4 check boxes in 2 side-by-side columns. Likewise if max is 3 and there are 9 check boxes, then this methods will create 3 columns of 3 check boxes. If this argument is omitted, all the check boxes are placed in a single column.

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces. Each check box in the group is given the style specified by this option.

3STATE	TOP	LTEXT
OWNER	BOTTOM	RBUTTON
BITMAP	VCENTER	HIDDEN
ICON	MULTILINE	DISABLED
LEFT	NOTIFY	NOBORDER
RIGHT	PUSHLIKE	GROUP
HCENTER	FLAT	NOTAB

3STATE

A three-state check box is created. All check box controls have two states, checked and cleared (not checked.) A three-state check box also has an indeterminate state. This is represented by a grayed box inside the check box. The programmer can use methods of the *CheckBox* class to determine which of the 3 states a check box is in.

OWNER

The programmer is completely responsible for drawing the button when the dialog receives the WM_DRAWITEM message. Currently this would be difficult (but not impossible) to implement in ooDialog. For a check box, the difficulty would probably make this impractical.

ICON

The button displays an icon image.

BITMAP

The button displays a bitmap image.

LEFT

Left justifies the text to the right of the check box.

RIGHT

Right justifies the text to the right side of the check box.

HCENTER

The text is centered horizontally to the right of the check box. This is the default if neither LEFT nor RIGHT are specified.

TOP

The text is aligned at the top of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

BOTTOM

The text is aligned at the bottom of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

VCENTER
: The text is vertically centered in the button rectangle. This is the default if neither TOP nor BOTTOM are specified.

MULTILINE
: If the text for the label of the button is longer than the width of the button rectangle, the operating system will word wrap the text producing more than one line. The height of the button rectangle has to be sufficient to display the extra line(s) or the text is clipped.

NOTIFY
: Enables the button to send notifications for the gained focus,lost focus, and double click events. This is only necessary when the *connectButtonEvent* method is used, and only for the GOTFOCUS, LOSTFOCUS, or DBLCLK keywords of that method. Note that only radio buttons and owner-drawn buttons will receive the DBLCLICK event.

PUSHLIKE
: Makes the button look and act like a push button. The button looks raised when it isn't pushed or checked, and sunken when it is pushed or checked. The button is drawn with a flat appearance, i.e., it is drawn as a flat rectangle the label text inside the rectangle.

FLAT
: The button is drawn with a flat appearance, i.e., it is drawn as a flat rectangle the label text inside the rectangle.

LTEXT
: Places text on the left side of the check box. This is the same as the RBUTTON style.

RBUTTON
: Positions the check box's square on the right side of the button rectangle. This is the same as the LTEXT style.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

NOBORDER
: Use this keyword to specify that a group *GroupBox* is not added around the check box group.

GROUP
: Do not use this keyword with this method. The group style is added correctly internally.

NOTAB
: The no *tabstop* control style.

idStat [optional]
: This argument is used to set the *resource ID* for the group *GroupBox*, if one is used. The argument is ignored if the NOBORDER style is used. If this argument is omitted and the NOBORDER style is not used, then -1 is assigned as the resource ID for the group box.

Example:

The following example is complete. It can be cut and pasted into a file and run as is. It adds a group of six check boxes with consecutive IDs of 304 through 309. They are placed in a group with two check boxes each in three columns. The group box for the group is give a resource ID of 300. In initDialog(), the group box is then given a label and the first and fifth check boxes are checked.

```
/* Simple test of create ... methods */

  dlg = .SimpleCB~new

  if dlg~initCode = 0 then do
    dlg~createCenter(215, 83, "Testing Check Boxes", "VISIBLE", , "Ms Shell Dlg 2", 8)
    dlg~execute("SHOWTOP")
  end

::requires "ooDialog.cls"

::class 'SimpleCB' subclass UserDialog

::method defineDialog

  labels.1 = 'Upper class'
  labels.2 = 'Middle class'
  labels.3 = 'Business class'
  labels.4 = 'Lower class'
  labels.5 = 'Classless'
  labels.6 = 'Class clown'
  self~createCheckBoxStem(304, 10, 13, ,labels., 2, "LEFTTEXT RIGHT", 300)

  self~createPushButton(IDOK, 10, 60, 50, 14, "DEFAULT GROUP", "OK")
  self~createPushButton(IDCANCEL, 65, 60, 50, 14, , "Cancel")

::method initDialog
  self~newGroupBox(300)~style = "LEFT"
  self~newGroupBox(300)~setText("Class Warfare")
  self~newCheckBox(304)~check
  self~newCheckBox(308)~check

::method initAutoDetection
  self~noAutoDetection
```

8.6.3.5. createGroupbox

```
>>--createGroupbox(-+------+-,-x-,-y-,-cx-,-cy--+---------+-+--------+-)-------><
                    +--id--+                    +-,-style-+ +-,-text-+
```

The *createGroupbox* method creates a group *GroupBox* in the dialog template.

Arguments:

This method takes the following arguments.

id [optional]

The *resource ID* for the group box. If omitted, -1 is used for the resource id. Group boxes normally are give -1 for their resource id because there is usually no need for the programmer to interact with a group box. However, if the programmer wants to modify something in the underlying group box, say to change its size, change its text, etc., then a positive resource id is needed.

x, y, cx, cy [required]

The control *coordinates*.

style [optional]

A list of 0 or more *style* keywords separated by spaces:

LEFT	HIDDEN	BORDER
RIGHT	DISABLED	TAB
CENTER	GROUP	

LEFT

The text is aligned to the left in the top edge of the group box frame. This is the default and does not need to be specified.

RIGHT

Normally the text is aligned to the upper left of the group box frame. This style aligns the text to the upper right of the frame. Note that aligning text to the right in a group box is rarely, if ever, done.

CENTER

This style aligns the text in the center of the line that forms the top of the group box frame. Normally the text is aligned to the upper left of the group box frame. Note that aligning text in the center of a group box is rarely, if ever, done.

HIDDEN

The not *visible* window style.

DISABLED

The *disabled* window style.

BORDER

The *border* window style.

GROUP

The *group* control style.

TAB

The *tabstop* control style.

text [optional]

The text for the group box label.

Return value:

The possible return values are:

-2

The dialog template has either not been started or is complete.

-1

A symbolic ID was used, but it could not be resolved correctly.

0

Success.

8.6.3.6. createOkCancelLeftBottom

```
>>--createOkCancelLeftBottom---------------------><
```

The *createOkCancelLeftBottom* method adds an OK and a Cancel push button group to the dialog template. The group is placed in a row at the lower-left edge of the dialog.

Arguments:

There are no arguments for this method.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

8.6.3.7. createOkCancelLeftTop

```
>>--createOkCancelLeftTop------------------------><
```

The *createOkCancelLeftTop* method creates an OK and a Cancel push button group in the dialog template. The group is placed in a column at the upper-left edge of the dialog.

Arguments:

There are no arguments for this method.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

8.6.3.8. createOkCancelRightBottom

```
>>--createOkCancelRightBottom--------------------><
```

The *createOkCancelRightBottom* method creates an OK and a Cancel push button group in the dialog template. The group is placed in a row at the lower-right edge of the dialog.

Arguments:

There are no arguments for this method.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

Example:

The following example adds the push buttons to the bottom of the dialog. It also overrides the standard OK and Cancel methods.

```
::class MyClass subclass UserDialog

::method defineDialog
      .
      .
      .
   self~createOkCancelRightBottom

::method OK
   ret = MessageBox("Are you sure?", "Please confirm", "OK")
   if ret=1 then self~OK:super

::method cancel
   ret = MessageBox("Do you really want to quit?", "Please confirm", "OK")
   if ret=1 then self~CANCEL:super
```

8.6.3.9. createOkCancelRightTop

```
>>--createOkCancelRightTop-----------------------><
```

The *createOkCancelRightTop* method creates an OK and a Cancel push button in the dialog template. The group is placed in a column at the upper-right edge of the dialog.

Arguments:

There are no arguments for this method.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

8.6.3.10. createPushButton

```
>>--createPushButton(-id-,-x-,-y-,-cx-,-cy-,-+---------+-+-------+-+-------+-)-><
                                             +-,-style-+ +-,-txt-+ +-,-mth-+
```

The *createPushButton* method creates a push *Button* in the dialog template and optionally connects it with a method that is invoked whenever the button is clicked.

Arguments:

The arguments are:

id [required]

The *resource ID* for the button.

x, y, cx, cy [required]

The control *coordinates*

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces:

DEFAULT	HCENTER	FLAT
OWNER	TOP	HIDDEN
BITMAP	BOTTOM	DISABLED
ICON	VCENTER	BORDER
LEFT	MULTILINE	GROUP
RIGHT	NOTIFY	NOTAB

DEFAULT

The button becomes the default button in the dialog.

OWNER

The programmer is completely responsible for drawing the button when the dialog receives the WM_DRAWITEM message. Currently this would be difficult (but not impossible) to implement in ooDialog. A simpler approach for an owner drawn button is to use an image list with the button. See the *setImageList*() method.

ICON

The button displays an icon image.

BITMAP

The button displays a bitmap image.

LEFT

Left justifies the text in the button rectangle. If neither LEFT nor RIGHT are specified the text is centered in the button rectangle.

RIGHT

Right justifies the text in the button rectangle. If neither LEFT nor RIGHT are specified the text is centered in the button rectangle.

HCENTER

The button has its text centered horizontally in the button rectangle. This is the default if neither LEFT nor RIGHT are specified.

TOP

The text is aligned at the top of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

BOTTOM

The text is aligned at the bottom of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

VCENTER

The text is vertically centered in the button rectangle. This is the default if neither TOP nor BOTTOM are specified.

MULTILINE

If the text for the label of the button is longer than the width of the button rectangle, the operating system will word wrap the text producing more than one line. The height of the button rectangle has to be sufficient to display the extra line(s) or the text is clipped.

NOTIFY

Enables the button to send notifications for the gained focus,lost focus, and double click events. This is only necessary when the *connectButtonEvent* method is used, and only for the GOTFOCUS, LOSTFOCUS, or DBLCLK keywords of that method.

FLAT

The button is drawn with a flat appearance, i.e., it is drawn as a flat rectangle the label text inside the rectangle.

HIDDEN

The not *visible* window style.

DISABLED

The *disabled* window style.

BORDER

The *border* window style.

GROUP

The *group* control style.

NOTAB

The no *tabstop* control style.

txt [optional]

The text for the label of the button. The default is the empty string.

mth [optional]

The name of a method of the Rexx dialog that is to be invoked each time the button is clicked. This serves the same purpose as the *connectButtonEvent* method for the CLICKED event.

Return value:

The possible return values are:

-2

The dialog template has either not been started or is complete.

-1

A symbolic ID was used, but it could not be resolved correctly.

0

Success.

Example:

The following example creates a push button with a label of "Get new Info" at position x=100, y=80 and size width=40, height=15. The button's ID is 555, and if the button is clicked, the *getInfo* method in the Rexx dialog is invoked. The button is created without the tabstop style. If the user navigates the dialog using the keyboard, the tab key will skip over the button.

```
MyDialog~createPushButton(555, 100, 80, 40, 15, "NOTAB", "&Get new Info", "getInfo")
```

8.6.3.11. createPushButtonGroup

```
                                                      +-----------+
                                                      V           |
>>--createPushButtonGroup(-x-,-y-+------+-+------+-,-txt- -id- -m-+--+-----+-+-----+-)-><
                                 +-,-cx-+ +-,-cy-+                   +-,-r-+ +-,-o-+
```

Use the *createPushButtonGroup* method to create more than one push button in the dialog template at one time. The buttons are arranged in a row or in a column.

Arguments:

The arguments are:

x, y [required]

The *coordinates* of the position of the upper left corner of the button group.

cx, cy [optional]

The width and height *coordinates* of a single button. Both cx and cy are optional. The default for cx is 40 and for cy 12.

txt ID m [required]

A single string containing *txt ID m* triplets (separated by blanks) for each button.

The first part of the triplet is the text that is displayed on the button. This must either be a single word, or, if the text is more than 1 word, the words must be enclosed in quotes within the string. The second part of the triplet is the *resource ID* of the button. This is of course a single word or number. The third part of the triplet is the name of a method in the Rexx dialog object that is connected to the button clicked *event*. This is also a single word.

The *txt id m* triplet can repeat as many times as desired, and determines how many buttons are created, one button for each triplet.

r [optional]

This is a flag to switch between horizontal or vertical placement of the buttons. If *r* is true, the buttons are placed in a row, or horizontal. If *r* is false, the buttons are placed in a column, or vertically.

o [optional]

A list of 0 or more of the following *style* keywords separated by spaces. Each button is given the same set of styles, except for the DEFAULT style. If the style arg contains the DEFAULT keyword, it is only given to the first button:

DEFAULT	HCENTER	FLAT
OWNER	TOP	HIDDEN
BITMAP	BOTTOM	DISABLED
ICON	VCENTER	BORDER
LEFT	MULTILINE	GROUP
RIGHT	NOTIFY	NOTAB

DEFAULT

The button becomes the default button in the dialog.

OWNER

The programmer is completely responsible for drawing the button when the dialog receives the WM_DRAWITEM message. Currently this would be difficult (but not impossible) to implement in ooDialog. A simpler approach for an owner drawn button is to use a bitmap button. See the *createBitmapButton* method.

ICON
: The button displays an icon image.

BITMAP
: The button displays a bitmap image.

LEFT
: Left justifies the text in the button rectangle. If neither LEFT nor RIGHT are specified the text is centered in the button rectangle.

RIGHT
: Right justifies the text in the button rectangle. If neither LEFT nor RIGHT are specified the text is centered in the button rectangle.

HCENTER
: The button has its text centered horizontally in the button rectangle. This is the default if neither LEFT nor RIGHT are specified.

TOP
: The text is aligned at the top of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

BOTTOM
: The text is aligned at the bottom of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

VCENTER
: The text is vertically centered in the button rectangle. This is the default if neither TOP nor BOTTOM are specified.

MULTILINE
: If the text for the label of the button is longer than the width of the button rectangle, the operating system will word wrap the text producing more than one line. The height of the button rectangle has to be sufficient to display the extra line(s) or the text is clipped.

NOTIFY
: Enables the button to send notifications for the gained focus,lost focus, and double click events. This is only necessary when the *connectButtonEvent* method is used, and only for the GOTFOCUS, LOSTFOCUS, or DBLCLK keywords of that method. Note that only radio buttons and owner-drawn buttons will receive the DBLCLK event.

FLAT
: The button is drawn with a flat appearance, i.e., it is drawn as a flat rectangle the label text inside the rectangle.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

BORDER
: The *border* window style.

GROUP
: The *group* control style.

NOTAB
: The no *tabstop* control style.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

Example:

The following example creates three buttons (Add Record, Delete Record, and Update Record.) The buttons are placed in a column alongside the right edge of the dialog. The upper left corner of the button group is at 20, 235. Note that the multi-word text labels for the buttons are enclosed in single quotes within the entire string

```
self~createPushButtonGroup(20, 235, , ,                         -
                          "'Add Record' 301 AddItem "          -
                          "'&Delete Record' 302 DeleteItem "   -
                          "'&Update Record' 303 UpdateItem")
```

8.6.3.12. createPushButtonStem

```
>>--createPushButtonStem(-+---+-+-----+-+------+-+------+-,-inp.-+-----+-+---------+-)-><
                          +-x-+ +-,-y-+ +-,-cx-+ +-,-cy-+        +-,-r-+ +-,-align-+
```

Use the *createPushButtonStem* method to create more than one push button in the dialog template at one time, using a **`Stem`** to input the individual button details. The buttons are arranged in a row or in a column.

Arguments:

The arguments are:

x, y [optional]
: The *coordinates* of the position of the upper left corner of the button group. If either x or y is omitted, then the position of the upper left corner is calculated. The calculated position takes into account the size of the buttons along with the *r* and *align* arguments. See the remarks section for details.

cx, cy [optional]
: The width and height *coordinates* of a single button. Both cx and cy are optional. If either cx or cy is omitted, then both cx and cy are calculated. The calculation is based on the size taken up by the largest label in all the buttons.

  The Microsoft documentation specifies the minimal size for a push button to be 50 dialog units for cx and 14 dialog units for cy. When cx and cy are calculated they are never set smaller than the minimum recommended size. This implies that to have smaller buttons than the minimum, the programmer must specify both cx and cy.

inp. [required]

A **`Stem`** whose indexes specify the details for each button. The possible indexes are as follows. Note that some indexes are required and some are optional. The stem indexes are structured in a way where **`inp.0`** specifies the number of buttons and **`inp.1`** handles the data for button one, **`inp.2`** handles the data for button two, up to **`inp.0`** buttons.

Note that there are two possible ways to construct the indexes, the tails, that specify the data for each button. One way is to explicity name the tails, like this:

```
inp.0 = 4
inp.1.id     = IDC_PB_PUSHME
inp.1.text   = "Push"
inp.1.method = onPushMe
inp.2.id     = IDOK
inp.2.text   = "Ok"
inp.2.method = ok
inp.2.opts   = 'DEFAULT'
inp.3.id     = IDCANCEL
inp.3.text   = "Cancel"
inp.3.method = cancel
```

or the data for a single button could be specified as a stem itself, like the following:

```
inp.0 = 4
...
btn.id   = IDHELP
btn.text = 'Help'
inp.4    = btn.
```

The stem indexes are as follows:

0 [required]

The value of this index must be a non-negative whole number that specifies the number of buttons in the group.

.ID [required]

This index specifies the *resource ID* of the button. This index must be specified for each button.

.TEXT [optional]

This index specifies the text, the label, of the button. If this index is missing, the empty string is used for the button.

.METHOD [optional]

This index specifies the name of a method in the Rexx dialog object that is connected to the button CLICKED *event*. If this index is missing, then no method is connected to the CLICKED event.

.OPTS [optional]

This index specifies the *style* for the button. The value of the index is a list of 0 or more of the following keywords separated by spaces. If this index is omitted for any button, the empty string is used:

DEFAULT	HCENTER	FLAT
OWNER	TOP	HIDDEN

BITMAP	BOTTOM	DISABLED
ICON	VCENTER	BORDER
LEFT	MULTILINE	GROUP
RIGHT	NOTIFY	NOTAB

DEFAULT

The button becomes the default button in the dialog.

OWNER

The programmer is completely responsible for drawing the button when the dialog receives the WM_DRAWITEM message. Currently this would be difficult (but not impossible) to implement in ooDialog. A simpler approach for an owner drawn button is to use a bitmap button. See the *createBitmapButton* method.

ICON

The button displays an icon image.

BITMAP

The button displays a bitmap image.

LEFT

Left justifies the text in the button rectangle. If neither LEFT nor RIGHT are specified the text is centered in the button rectangle.

RIGHT

Right justifies the text in the button rectangle. If neither LEFT nor RIGHT are specified the text is centered in the button rectangle.

HCENTER

The button has its text centered horizontally in the button rectangle. This is the default if neither LEFT nor RIGHT are specified.

TOP

The text is aligned at the top of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

BOTTOM

The text is aligned at the bottom of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

VCENTER

The text is vertically centered in the button rectangle. This is the default if neither TOP nor BOTTOM are specified.

MULTILINE

If the text for the label of the button is longer than the width of the button rectangle, the operating system will word wrap the text producing more than one line. The height of the button rectangle has to be sufficient to display the extra line(s) or the text is clipped.

NOTIFY

Enables the button to send notifications for the gained focus,lost focus, and double click events. This is only necessary when the *connectButtonEvent* method is used, and only for the GOTFOCUS, LOSTFOCUS, or DBLCLK keywords of that method. Note that only radio buttons and owner-drawn buttons will receive the DBLCLK event.

FLAT
: The button is drawn with a flat appearance, i.e., it is drawn as a flat rectangle the label text inside the rectangle.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

BORDER
: The *border* window style.

GROUP
: The *group* control style.

NOTAB
: The no *tabstop* control style.

r [optional]
: True or false to switch between horizontal or vertical placement of the buttons. If *r* is true, the buttons are placed in a row, or horizontally. If *r* is false, the buttons are placed in a column, or vertically. If *r* is omitted, the default is true.

align [optional]
: By default, when the button positioning is calculate, the button row or column is aligned so that the bottom right corner of the is at the lower right margin of the dialog. Note that Microsoft recommends that all dialogs have a margin of 7 dialog units.

  This default can be changed using the *align* string argument. If the string contains the keyword TOP, then the default is to align the buttons at the top margin. If the sting contains the keyword LEFT, then the default is to align the button at the left margin. Both keywords are case insensitive and can be used separately or together. Any other words in the string are ignored. However, when both the *x* and *y* arguments are specified, the *align* argument is ignored completely.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete. Or the *inp.* stem was not formed correctly.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

1
: Some other error occurred.

Remarks:

Example:

The following example creates three buttons (Add Record, Delete Record, and Update Record.) The buttons are placed in a column alongside the upper right margin of the dialog. Both the proper size for the buttons and the correct x, y coordinates of the button group are calculated:

```
inp.0 = 3
inp.1.id     = IDC_PB_ADD
inp.1.text   = "Add Record"
inp.1.method = addItem
inp.2.id     = IDC_PB_DEL
inp.2.text   = "Delete Record"
inp.2.method = deleteItem
inp.3.id     = IDC_PB_UPDATE
inp.3.text   = "Update Record"
inp.3.method = updateItem

self~createPushButtonStem( , , , , inp., .false, "TOP")
```

8.6.3.13. createRadioButton

```
>>--createRadioButton(-id-,-x-,-y-+------+--+------+--+---------+-------------->
                                  +-,-cx-+  +-,-cy-+  +-,-style-+

>--+--------+--+--------+--+-------------+--)--------------------------------><
   +-,-text-+  +-,-attr-+  +-,-connectMth-+
```

The *createRadionButton* method adds a radio button to the dialog template. The radio button is created as an automatic radio *RadioButton*. There is no way using this method to create a standard radio button.

Arguments:

The arguments are:

id [required]

The *resource ID* for the control.

x [required], y [required], cx [optional], cy [optional]

The control *coordinates*. If either cx or cy are omitted, then the size for the radio button is calculated internally using its label. (The text argument.)

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces:

OWNER	BOTTOM	RBUTTON
BITMAP	VCENTER	HIDDEN
ICON	MULTILINE	DISABLED
LEFT	NOTIFY	BORDER
RIGHT	PUSHLIKE	GROUP
HCENTER	FLAT	NOTAB
TOP	LTEXT	

OWNER

The programmer is completely responsible for drawing the button when the dialog receives the WM_DRAWITEM message. Currently this would be difficult (but not impossible) to implement in ooDialog. For a radio button, the difficulty would probably make this impractical.

ICON
: The button displays an icon image.

BITMAP
: The button displays a bitmap image.

LEFT
: Left justifies the text to the right of the radio button.

RIGHT
: Right justifies the text to the right side of the radio button.

HCENTER
: The text is centered horizontally to the right of the radio button. This is the default if neither LEFT nor RIGHT are specified.

TOP
: The text is aligned at the top of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

BOTTOM
: The text is aligned at the bottom of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

VCENTER
: The text is vertically centered in the button rectangle. This is the default if neither TOP nor BOTTOM are specified.

MULTILINE
: If the text for the label of the button is longer than the width of the button rectangle, the operating system will word wrap the text producing more than one line. The height of the button rectangle has to be sufficient to display the extra line(s) or the text is clipped.

NOTIFY
: Enables the button to send notifications for the gained focus, lost focus, and double click events. This is only necessary when the *connectButtonEvent* method is used, and only for the GOTFOCUS, LOSTFOCUS, or DBLCLK keywords of that method. Note that only radio buttons and owner-drawn buttons will receive the DBLCLICK event.

PUSHLIKE
: Makes the button look and act like a push button. The button looks raised when it isn't pushed or checked, and sunken when it is pushed or checked. The button is drawn with a flat appearance, i.e., it is drawn as a flat rectangle the label text inside the rectangle.

FLAT
: The button is drawn with a flat appearance, i.e., it is drawn as a flat rectangle the label text inside the rectangle.

LTEXT
: Places text on the left side of the radio button. This is the same as the RBUTTON style.

RBUTTON
: Positions the radio button's circle on the right side of the button rectangle. This is the same as the LTEXT style.

HIDDEN
: The control has the not *visible* window style.

DISABLED
: The control has the *disabled* window style.

BORDER
: The control has the *border* window style.

GROUP
: The control has the *group* style. For the auto radio buttons to work, the first radio button should be given the group style. None of the other radio buttons should have the group style. The first control that has the group style then ends the group.

NOTAB
: The control does not have the *tabstop* style.

text [optional]
: The text for the label that is displayed next to the radio button. If this argument is omitted, the label will be the empty string.

attr [optional]
: The name of the data *attribute* associated with the radio button. If you omit this argument then the name of the data attribute is constructed *automatically*. If *automatic* data detection is off, this argument is ignored.

connectMth [optional]
: A text string that allows the *connectButtonEvent* CLICKED event notification to be automatically connected to a method in the dialog. In normal usage *connectMth* is the name of the method in the Rexx dialog to be connected to the CLICKED event.

However, the *createRadioButton* method is also used by the *RcDialog* to create radio button controls found in the resource *script* for the dialog. Therefore, if this string contains the word CONNECTRADIOS, case insignificant, then the method name is generated in the same way as it is by specifying CONNECTRADIOS in the *opts* argument when *new* a **`RcDialog`**.

First a string is constructed from the label of the button with all space, tab, ampersand, colon, and plus sign characters removed. In addition, if the label were to have a trailing ... that is also removed. Then that string is prepended with the text **`id`**. This string is used as the method name to connect the clicked event to.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

1
: The control was added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

Example:

The following example defines seven radio buttons with IDs 501 through 507:

```
RText.1="Monday"
RText.2="Tuesday"
RText.3="Wednesday"
RText.4="Thursday"
RText.5="Friday"
RText.6="Saturday"
RText.7="Sunday"

do i=1 to 7
 MyDialog~createRadioButton(500+i, 20, i*15+13, 40, 14, , RText.i)
end
```

Note: that there are also methods that create a whole group of radio buttons automatically. Such as the *createRadioButtonGroup* and *createRadioButtonStem* methods.

8.6.3.14. createRadioButtonGroup

```
                                                 +-----+
                                                 V     |
>>--createRadioButtonGroup(id1,-x-,-y-+------+-,---txt-+--+---------+-+----------+-)--><
                                      +-,-cx-+            +-,-style-+ +-,-idSt-+
```

Creates a complete group of auto radio buttons.

Arguments:

The arguments are:

id1 [required]

The *resource ID* for the first radio button. This id is increased by 1 for each additional radio button and then assigned to the control.

x [required], y [required], cx [optional]

The control *coordinates*. x and y name the position of the first radio button control. The other radio buttons are positioned automatically. cx specifies the length of the radio button plus text. If omitted, the space needed is calculated.

text [required]

The text string for each radio button. Single words have to be separated by blank spaces. This argument determines the number of radio buttons in total. Note that this prevents using more than one word for the label of any radio button.

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces. Each radio button in the group is given the style specified by this option.

OWNER	BOTTOM	RBUTTON
BITMAP	VCENTER	HIDDEN
ICON	MULTILINE	DISABLED
LEFT	NOTIFY	NOBORDER
RIGHT	PUSHLIKE	GROUP
HCENTER	FLAT	NOTAB
TOP	LTEXT	

OWNER
: The programmer is completely responsible for drawing the button when the dialog receives the WM_DRAWITEM message. Currently this would be difficult (but not impossible) to implement in ooDialog. For a radio button, the difficulty would probably make this impractical.

ICON
: The button displays an icon image.

BITMAP
: The button displays a bitmap image.

LEFT
: Left justifies the text to the right of the radio button.

RIGHT
: Right justifies the text to the right side of the radio button.

HCENTER
: The text is centered horizontally to the right of the radio button. This is the default if neither LEFT nor RIGHT are specified.

TOP
: The text is aligned at the top of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

BOTTOM
: The text is aligned at the bottom of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

VCENTER
: The text is vertically centered in the button rectangle. This is the default if neither TOP nor BOTTOM are specified.

MULTILINE
: If the text for the label of the button is longer than the width of the button rectangle, the operating system will word wrap the text producing more than one line. The height of the button rectangle has to be sufficient to display the extra line(s) or the text is clipped.

NOTIFY
: Enables the button to send notifications for the gained focus,lost focus, and double click events. This is only necessary when the *connectButtonEvent* method is used, and only for the GOTFOCUS, LOSTFOCUS, or DBLCLK keywords of that method. Note that only radio buttons and owner-drawn buttons will receive the DBLCLICK event.

PUSHLIKE
: Makes the button look and act like a push button. The button looks raised when it isn't pushed or checked, and sunken when it is pushed or checked. The button is drawn with a flat appearance, i.e., it is drawn as a flat rectangle the label text inside the rectangle.

FLAT
: The button is drawn with a flat appearance, i.e., it is drawn as a flat rectangle the label text inside the rectangle.

LTEXT
: Places text on the left side of the radio button. This is the same as the RBUTTON style.

RBUTTON
: Positions the radio button's circle on the right side of the button rectangle. This is the same as the LTEXT style.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

NOBORDER
: Use this keyword to specify that a group *GroupBox* is not added around the radio button group.

GROUP
: Do not use this keyword with this method. The group style is added correctly internally.

NOTAB
: The no *tabstop* control style.

idstat [optional]
: This argument is used to set the *resource ID* for the group *GroupBox*, if one is used. The argument is ignored if the NOBORDER style is used. If omitted and the NOBORDER style is not used, then -1 is used for the resource ID.

Example:

The following example adds a group of three radio buttons with IDs 301, 302, and 303 to the *dialog*:

```
MyDialog = .UserDialog~new
MyDialog~create(100, 100, 80, 60, "Radio Button Group")
MyDialog~createRadioButtonGroup(301, 23, 18, ,"Fast Medium Slow")
MyDialog~fast = 1
MyDialog~execute
```

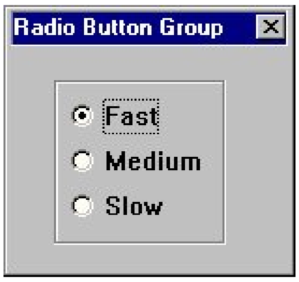


Figure 8.2. Sample Radio Button Group

8.6.3.15. createRadioButtonStem

```
>>--createRadioButtonStem(-id-,-x-,-y--+------+-,-text.--+-------+------------->
                                       +-,-cx-+          +-,-max-+

>--+---------+--+----------+-)------------------------------------------------><
```

```
    +-,-style-+  +-,-idStat-+
```

The *createRadioButtonStem* method adds a group of radio button controls to the dialog. It is very similar to the *createRadioButtonGroup* method. It simply uses a stem to specify the text label for each radio button. Notice that this will allow the labels to consist of more than one word. This removes the restriction of the **`createRadioButtonGroup`** method that labels must be a single word.

Arguments:

The arguments are:

id [required]

The *resource ID* for the first radio button. This id is increased by 1 for each additional radio button and then assigned to the control.

x, y [required]

The control position *coordinates*. x and y name the position of the first radio button control. The other radio buttons are positioned automatically.

cx [optional]

The control width *coordinates*. cx specifies the length of the radio button plus text. If omitted, the space needed is calculated.

text. [required]

A stem containing the text label for each radio button. The stem indexes should start at 1 and continue consecutively, with each succeeding number containing the label for the next radion button. This argument determines the number of radio buttons in total.

max [optional]

A number specifying the maximum radio buttons to put in a column. The programmer can use this to balance the look of the group by placing the buttons in more than 1 column. For instance, if there are 8 radio buttons and the max argument is 4 then the result will have 4 radio buttons in 2 side-by-side columns. If this argument is omitted then all radio buttons are all put in 1 column.

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces. Each radio button in the group is given the style specified by this option.

OWNER	BOTTOM	RBUTTON
BITMAP	VCENTER	HIDDEN
ICON	MULTILINE	DISABLED
LEFT	NOTIFY	NOBORDER
RIGHT	PUSHLIKE	GROUP
HCENTER	FLAT	NOTAB
TOP	LTEXT	

OWNER

The programmer is completely responsible for drawing the button when the dialog receives the WM_DRAWITEM message. Currently this would be difficult (but not impossible) to implement in ooDialog. For a radio button, the difficulty would probably make this impractical.

ICON

The button displays an icon image.

BITMAP

The button displays a bitmap image.

LEFT
: Left justifies the text to the right of the radio button.

RIGHT
: Right justifies the text to the right side of the radio button.

HCENTER
: The text is centered horizontally to the right of the radio button. This is the default if neither LEFT nor RIGHT are specified.

TOP
: The text is aligned at the top of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

BOTTOM
: The text is aligned at the bottom of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

VCENTER
: The text is vertically centered in the button rectangle. This is the default if neither TOP nor BOTTOM are specified.

MULTILINE
: If the text for the label of the button is longer than the width of the button rectangle, the operating system will word wrap the text producing more than one line. The height of the button rectangle has to be sufficient to display the extra line(s) or the text is clipped.

NOTIFY
: Enables the button to send notifications for the gained focus,lost focus, and double click events. This is only necessary when the *connectButtonEvent* method is used, and only for the GOTFOCUS, LOSTFOCUS, or DBLCLK keywords of that method. Note that only radio buttons and owner-drawn buttons will receive the DBLCLICK event.

PUSHLIKE
: Makes the button look and act like a push button. The button looks raised when it isn't pushed or checked, and sunken when it is pushed or checked. The button is drawn with a flat appearance, i.e., it is drawn as a flat rectangle the label text inside the rectangle.

FLAT
: The button is drawn with a flat appearance, i.e., it is drawn as a flat rectangle the label text inside the rectangle.

LTEXT
: Places text on the left side of the radio button. This is the same as the RBUTTON style.

RBUTTON
: Positions the radio button's circle on the right side of the button rectangle. This is the same as the LTEXT style.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

NOBORDER
: Use this keyword to specify that a group *GroupBox* is not added around the radio button group.

GROUP
: Do not use this keyword with this method. The group style is added correctly internally.

NOTAB
: The no *tabstop* control style.

idstat [optional]
: This argument is used to set the *resource ID* for the group *GroupBox*, if one is used. The argument is ignored if the NOBORDER style is used. If this argument is omitted and the NOBORDER style is not used then -1 is used for the resource ID.

Details:

The documentation prior to version 4.0.0 listed a font name and a font size argument. These arguments do nothing and are ignored in ooDialog 4.0.0 and later.

Example:

The following example is complete. It can be cut and pasted into a file and run as is. It adds a group of four radio buttons with IDs 304, 305, 306, and 307, in a group with two buttons each in two columns. The group box for the group is give a resource ID of 300. In initDialog(), the group box is then given a label and the first radio button is checked.

```
/* Simple test of create ... */

  dlg = .SimpleRB~new

  if dlg~initCode = 0 then do
    dlg~createCenter(150, 83, "Testing Radio Buttons", "VISIBLE", , "Ms Shell Dlg 2", 8)
    dlg~execute("SHOWTOP")
  end

::requires "ooDialog.cls"

::class 'SimpleRB' subclass UserDialog

::method defineDialog

  labels.1 = 'Upper class'
  labels.2 = 'Middle class'
  labels.3 = 'Business class'
  labels.4 = 'Lower class'
  self~createRadioButtonStem(304, 10, 13, ,labels., 2, "LEFTTEXT", 300)

  self~createPushButton(IDOK, 10, 60, 50, 14, "DEFAULT GROUP", "OK")
  self~createPushButton(IDCANCEL, 65, 60, 50, 14, , "Cancel")

::method initDialog
  self~newGroupBox(300)~setTitle("Class Warfare")
  self~newRadioButton(304)~check
```

8.6.4. Create Combo Box Controls

The methods in this section are all used to create combo *ComboBox* controls in the dialog *template*.

8.6.4.1. createComboBox

```
>>--createComboBox(--id--,--x--,--y--,--cx--,--cy--+---------+--+------------+--)-><
                                                   +-,-style-+  +-,-attrName-+
```

The *createComboBox* method creates a combo *ComboBox* in the dialog template.

Arguments:

The arguments are:

id [required]

The *resource ID* for the control.

x, y, cx, cy [required]

The combo box *coordinates*.

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces:

SIMPLE	NOHSCROLL	HIDDEN
LIST	PARTIAL	GROUP
DISABLENOSCROLL	SORT	DISABLED
HSCROLL	VSCROLL	NOBORDER
LOWER	UPPER	NOTAB

SIMPLE

Adds a simple combo box, which displays the list box at all times. If neither SIMPLE nor LIST are specified, the combo box will be a drop-down combo box.

LIST

Adds a drop-down list combo box. If neither SIMPLE nor LIST are specified, the combo box will be a drop-down combo box.

DISABLENOSCROLL

When the vertical height of the items in the list box is less than the height of the list box, the vertical scroll bar is shown as disabled rather than hidden. Without this style, the scroll bar is hidden when the vertical height of the items in the list box is less than the height of the list box.

Horizontal scroll bars are effected in a similar manner. If the width of the list box is equal to or greater than the horizontal extent horizontal extent, then the horizontal scroll bar is disabled rather than hidden.

This style has no effect unless scroll bars have been added using the VSCROLL or HSCROLL styles.

HSCROLL

Adds a horizontal scroll bar to the combo box. If the width of the list box is smaller than the horizontal extent, then the horizontal scroll bar can be used to horizontally scroll items in the list box. If the width of the list box is equal to or greater than the horizontal extent, the horizontal scroll bar is hidden, unless the combo box also has the DISABLENOSCROLL style.

When the DISABLENOSCROLL style is used, the scroll bar will be disabled rather than hidden. In addition, when the DISABLENOSCROLL style is used, a horizontal scroll bar will be added without specifying the HSCROLL style.

Note that the horizontal extent for the combo box is 0 when the combo box is created. Because of that, adding a horizontal scroll will have no effect unless the programmer also changes the extent using the *setHorizontalExtent*.

LOWER
Converts all the text in the list and the selection field to lowercase.

NOHSCROLL
In combo boxes where the *ComboBox* field is an edit control, the combo box normally scrolls the text automatically to the right when the user types a character at the end of the line. When the NOHSCROLL style is set, automatic horizontal scrolling is disabled and the user can not enter more text than fits within the width of the selection field.

PARTIAL
Specifies that the size of the combo box is exactly the size specified by the application when it created the combo box. Normally, the system sizes a combo box so that it does not display partial items.

SORT
The items in the list are sorted by the combo box itself.

UPPER
Converts the text in the list and the selection field to uppercase.

VSCROLL
Adds a vertical scroll bar to the combo box. When the vertical height of the items in the combo box is greater than the height of the list box of the combo box, then the scroll bar will scroll the items vertically.

Note that without a vertical scroll bar, the user can still use the up and down arrows to scroll through all of the items in the list box. The vertical scroll bar is not necessary. When the vertical height of the items in the combo box is less than the height of the list box, the scroll bar is hidden, or disabled if the combo box has the DISABLENOSCROLL style.

HIDDEN
Create the combo box without the *visible* style.

DISABLED
Create the combo box with the *disabled* style.

GROUP
Create the combo box with the *group* style.

NOTAB
Create the combo box without the *tabstop* style. By default the combo box is created with the tabstop style.

NOBORDER
Create the combo box without the *border* style. By default combo boxes are created with a border.

attrName [optional]

The name of the data *attribute* associated with the combo box control. If you omit this argument then the name of the data attribute is constructed *automatically*. If *automatic* data detection is off, this argument is ignored.

Return value:

The possible return values are:

-2

The dialog template has either not been started or is complete.

-1

A symbolic ID was used, but it could not be resolved correctly.

0

Success.

1

The control was added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

8.6.4.2. createComboBoxInput

```
>>--createComboBoxInput(-id-,-x-,-y-+-------+-,-cx2-+---------+-,--text-------->
                                    +-,-cx1-+       +-,-count-+

>---+---------+---+----------+--+----------+--)------------------------------><
    +-,-style-+   +-,-idStat-+  +-,-attName-+
```

The *createComboBoxInput* method is used to create a combo box and a static text control in the dialog template. The static text controls is used to provide a label for the combo box.

Arguments:

The arguments are:

id [required]

The *resource ID* for the control.

x, y [required]

The position *coordinates* of the input group.

cx1 [optional]

The width *coordinates* of the static text control used for the label. If omitted, the width is calculated automatically using the dialog font. The height of the label is always calculated automatically using the dialog font.

cx2 [required]

The width *coordinates* of the combo box.

count [optional]

The height *coordinates* of the combo box specified as a number of lines. The height of one line if text is calculated using the dialog font. This height is used for the height of the label. The *count* argument then specifies how many lines. of that height, the combo box should be. The default is 5 lines.

text [required]
: The text for the label. The label is displayed on the left-hand side of the combo box.

style [optional]
: A list of 0 or more of the following *style* keywords separated by spaces:

SIMPLE	NOHSCROLL	DISABLED
LIST	PARTIAL	NOBORDER
SORT	HIDDEN	NOTAB
VSCROLL	GROUP	

SIMPLE
: Adds a simple combo box, which displays the list box at all times. If neither SIMPLE nor LIST are specified, the combo box will be a drop-down combo box.

LIST
: Adds a drop-down list combo box. If neither SIMPLE nor LIST are specified, the combo box will be a drop-down combo box.

SORT
: The items in the list are sorted by the combo box itself.

VSCROLL
: Adds a vertical scroll bar to the combo box.

NOHSCROLL
: In combo boxes where the *ComboBox* field is an edit control, the combo box normally scrolls the text automatically to the right when the user types a character at the end of the line. When the NOHSCROLL style is set, automatic horizontal scrolling is disabled and the user can not enter more text than fits within the width of the selection field.

PARTIAL
: Specifies that the size of the combo box is exactly the size specified by the application when it created the combo box. Normally, the system sizes a combo box so that it does not display partial items.

HIDDEN
: Create the combo box without the *visible* style.

DISABLED
: Create the combo box with the *disabled* style.

GROUP
: Create the combo box with the *group* style.

NOTAB
: Create the combo box without the *tabstop* style. By default the combo box is created with the tabstop style.

NOBORDER
: Create the combo box without the *border* style. By default combo boxes are created with a border.

idStat [optional]
: This argument is used to set the *resource ID* for the label, if one is desired. If omitted, the default static control ID (-1) is used.

attrName [optional]

The name of the data *attribute* associated with the combo box control. If you omit this argument then the name of the data attribute is constructed *automatically*. If *automatic* data detection is off, this argument is ignored.

Return value:

The possible return values are:

-2

The dialog template has either not been started or is complete.

-1

A symbolic ID was used, but it could not be resolved correctly.

0

Success.

1

The combobox control was added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

8.6.5. createDateTimePicker

```
>>--createDateTimePicker(-id--x-,--y-,--cx-,--cy-+---------+--+------------+--)-><
                                                  +-,-style-+  +-,-attrName-+
```

Creates a date *DateTimePicker* picker control in the dialog template and specifies the resource ID, position, size, style, and attribute name for the control.

Arguments:

The arguments are:

id [required]

The *resource ID* for the date time picker.

x, y, cx, cy [required]

The control *coordinates*

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces. **Note** that the LONG, SHORT, CENTURY, and TIME styles are mutually exclusive. If the programmer specifies more than one of those style keywords, only one of the styles will be in effect. Which style is in effect is undefined.

LONG	RIGHT	GROUP
SHORT	SHOWNONE	BORDER
CENTURY	UPDOWN	NOTAB
TIME	HIDDEN	
CANPARSE	DISABLED	

LONG

The date and time picker control will display the date in long format. This format will display the date like: "Sunday, December 26, 2010". Mutually exclusive with the SHORT, CENTURY, and TIME styles.

SHORT
: The control uses the short date format which will look like: "12/26/10". Mutually exclusive with the Long, CENTURY, and TIME styles

CENTURY
: The CENTURY style displays the date similarly to the SHORT style, except the year uses all four digits: "12/26/2010". Mutually exclusive with the LONG, SHORT, and TIME styles.

TIME
: The date and time picker displays the time. The format of the time is like: "3:39:03 PM".

CANPARSE
: This style allows the programmer to parse the user input and take an appropriate action. It allows a users to edit within the client area of the control by pressing the F2 key. The programmer uses the *connectDateTimePickerEvent* method to connect the USERSTRING *event* notification to the Rexx dialog object. The date and time picker control sends this notification message when the user is finished.

RIGHT
: The drop-down month calendar is right-aligned with the date and time picker. The default is for the drop-down to be left-aligned.

SHOWNONE
: This style makes it possible to have no date currently selected in the control. The date and time picker displays a check box that is selected automatically whenever a date is picked or entered. When the check box is deselected, the programmer can not get the date from the control, the date is essentially no date. The state of the check box can be set using the *setDateTime* method or queried by using the *getDateTime* method.

UPDOWN
: This style shows an *UpDown* control to the right of the date and time picker. The use can modify date and time values using the up-down control. This style can be used in place of the drop-down *MonthCalendar* calendar. The drop-down month calendar is the default style.

HIDDEN
: The date and time picker has the not *visible* window style.

DISABLED
: The date and time picker has the *disabled* window style.

BORDER
: The date and time picker has the *border* window style. By default a date and time picker is created without the border style.

GROUP
: The date and time picker has the *group* control style.

NOTAB
: The date and time picker is created without *tabstop* control style. The default is to create the date and time picker with the tabstop style.

attrName [optional]

The name of the data *attribute* associated with the dialog control. If you omit this argument then the name of the data attribute is constructed *automatically*. If *automatic* data detection is off, this argument is ignored.

Return value:

The possible return values are:

-2

The dialog template has either not been started or is complete.

-1

A symbolic ID was used, but it could not be resolved correctly.

0

Success.

1

The control was added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

Example:

The following example creates a date and time picker control with the *symbolic* ID of IDC_DT1 positioned at (17, 7) of the dialog's *client area*. The control will be 79 *dialog unit*s wide and 15 dialog units high. The date will be in the short date format.

```
self~createDateTimePicker(IDC_DT1, 17, 7, 79, 15, "SHORT")
```

8.6.6. Create Edit Controls

The methods in this section are all used to create *Edit* controls in the dialog *template*.

8.6.6.1. createEdit

```
>>--createEdit(-id-,-x-,-y-,-cx--+------+--+---------+--+------------+--)------><
                                 +-,-cy-+  +-,-style-+  +-,-attrName-+
```

Creates an *Edit* control in the dialog template and specifies the resource ID, position, size, style, and attribute name for the control.

Arguments:

The arguments are:

id [required]

The *resource ID* for the control.

x, y, cx [required]

The control position and width *coordinates*.

cy [optional]

The height *coordinates* for the control. If omitted, then the height is calculated automatically to fit the height of the dialog font.

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces:

Note: The behavior described here for the style keywords may vary slightly depending on the version of Windows.

AUTOSCROLLH	READONLY	NUMBER
HSCROLL	KEEPSELECTION	OEM
AUTOSCROLLV	PASSWORD	HIDDEN
VSCROLL	CENTER	DISABLED
MULTILINE	RIGHT	GROUP
HIDESELECTION	UPPER	NOTAB
NOWANTRETURN	LOWER	NOBORDER

AUTOSCROLLH

Automatically scrolls text to the right when the user types a character at the end of the line. In a single-line edit control text can not be added past the width of the control without this style. Multi-line edit controls do not need this style if a horizontal scroll bar is added.

HSCROLL

Adds a horizontal scroll bar to the bottom of the edit control. A horizontal scroll bar is only functional in multi-line edit controls.

AUTOSCROLLV

Automatically scrolls text up when the user presses ENTER on the last line. This style only affects multi-line edit controls and is not needed if a vertical scroll bar is added.

VSCROLL

Adds a vertical scroll bar to the right of the edit control.

MULTILINE

Designates a multiple-line edit control. (The default is single line.) Multi-line edit controls have the WANTRETURN and KEEPSELECTION styles unless the NOWANTRETURN and / or HIDESELECTION keywords are specified.

When a multi-line edit control has the WANTRETURN style, pressing the ENTER key inserts a line break in the text of the control. Without the WANTRETURN style, pressing the ENTER key clicks the default button. With the KEEPSELECTION style, any selected text in the edit control keeps its highlight when the control loses focus.

HIDESELECTION

Removes the KEEPSELECTION style from multi-line edit controls. This keyword only effects multi-line edit controls.

NOWANTRETURN

Removes the WANTRETURN style from multi-line edit controls. This keyword only effects multi-line edit controls.

READONLY

Prevents the user from entering or editing text in the edit control.

KEEPSELECTION

Selected text in an edit control keeps its highlight when the focus is shifted away from the control.

PASSWORD
: Characters typed into the edit control are masked. Rather than displaying the character typed, the edit control displays the mask character. Prior to Windows XP the default mask character was the asterisk. In Windows XP the default mask character is a black circle. The default password character can be changed through the *passwordChar* attribute of the *Edit* control.

CENTER
: Centers text in an edit control.

RIGHT
: Aligns text flush right in an edit control.

UPPER
: Converts all characters to uppercase as they are typed into the edit control.

LOWER
: Converts all characters to lowercase as they are typed into the edit control.

NUMBER
: Only allows digits to be typed into the edit control. If the user types an non-digit character an information balloon is displayed informing the user that only digits are allowed. This style does not prevent the user from pasting non-digits into the control.

OEM
: Converts text entered in the edit control from the Windows character set to the OEM character set and then back again to the Windows character set. This ensures proper character conversion in certain circumstances. Microsoft states that this style is most useful when edit controls contain file names that will be used with file systems that do not support Unicode. For further details see the Windows *documentation* provided by Microsoft.

HIDDEN
: Create the edit control without the *visible* style.

DISABLED
: Create the edit control with the *disabled* style.

GROUP
: Create the edit control with the *group* style.

NOTAB
: Create the edit control without the *tabstop* style. By default the edit control is created with the tabstop style.

NOBORDER
: Create the edit control without the *border* style. By default edit controls are created with a border.

attrName [optional]
: The name of the data *attribute* associated with the edit control. If you omit this argument then the name of the data attribute is constructed *automatically*. If *automatic* data detection is off, this argument is ignored.

Return value:

The possible return values are:

-2

The dialog template has either not been started or is complete.

-1

A symbolic ID was used, but it could not be resolved correctly.

0

Success.

1

The control was added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

Example:

The following example creates an edit control with ID 201 and length of 150 dialog units in the dialog template. It is positioned close to the upper-left corner of the dialog's client area. The FIRSTNAME attribute is created and connected to the dialog control. If the entered data is longer than 150 dialog units, the edit control scrolls horizontally.

```
dlg~createEdit(201, 12, 14, 150, ,"AUTOSCROLLH", "FIRSTNAME")
```

8.6.6.2. createEditInput

```
>>--createEditInput(-id-,-x-,-y-+-------+-,-cx2-,-+------+-,-text-+----------+->
                                +-,-cx1-+         +-,-cy-+         +-,-style-+

>--+------------+-+--------+-)-------------------------------------------------><
   +-,-idStatic-+ +-,-attr-+
```

The *createEditInput* method creates an edit control and a static text control (the label for the edit control) in the dialog template, in one step.

Arguments:

The arguments are:

id [required]

The *resource ID* for the control.

x, y, cx2 [required]

The position (x, y) *coordinates* of the pair of controls, along with the width coordinate of the edit control. x and y are the position of the upper-left edge of the label. The edit control is aligned automatically. cx2 is the width of the edit control.

cx1, cy [optional]

cx1 is the width *coordinates* of the static control. If omitted, this width is calculated by the ooDialog framework. cy is the height coordinates of the two controls. If it is omitted, it is also calculated automatically using the height of dialog font.

style [optional]

A list of 0 or more of the following *style* keywords, separated by spaces, used to set the style of the edit control.

Note: The behavior described here for the style keywords may vary slightly depending on the version of Windows.

AUTOSCROLLH	READONLY	NUMBER
HSCROLL	KEEPSELECTION	OEM
AUTOSCROLLV	PASSWORD	HIDDEN
VSCROLL	CENTER	DISABLED
MULTILINE	RIGHT	GROUP
HIDESELECTION	UPPER	NOTAB
NOWANTRETURN	LOWER	NOBORDER

AUTOSCROLLH
: Automatically scrolls text to the right when the user types a character at the end of the line. In a single-line edit control text can not be added past the width of the control without this style. Multi-line edit controls do not need this style if a horizontal scroll bar is added.

HSCROLL
: Adds a horizontal scroll bar to the bottom of the edit control. A horizontal scroll bar is only functional in multi-line edit controls.

AUTOSCROLLV
: Automatically scrolls text up when the user presses ENTER on the last line. This style only effects multi-line edit controls and is not needed if a vertical scroll bar is added.

VSCROLL
: Adds a vertical scroll bar to the right of the edit control.

MULTILINE
: Designates a multiple-line edit control. (The default is single line.) Multi-line edit controls have the WANTRETURN and KEEPSELECTION styles unless the NOWANTRETURN and / or HIDESELECTION keywords are specified.

: When a multi-line edit control has the WANTRETURN style, pressing the ENTER key inserts a line break in the text of the control. Without the WANTRETURN style, pressing the ENTER key clicks the default button. With the KEEPSELECTION style, any selected text in the edit control keeps its highlight when the control loses focus.

HIDESELECTION
: Removes the KEEPSELECTION style from multi-line edit controls. This keyword only effects multi-line edit controls.

NOWANTRETURN
: Removes the WANTRETURN style from multi-line edit controls. This keyword only effects multi-line edit controls.

READONLY
: Prevents the user from entering or editing text in the edit control.

KEEPSELECTION
: Selected text in an edit control keeps its highlight when the focus is shifted away from the control.

PASSWORD
: Characters typed into the edit control are masked. Rather than displaying the character typed, the edit control displays the mask character. Prior to Windows XP the default mask character was the asterisk. In Windows XP the default mask character is a black circle.

The default password character can be changed through the *passwordChar* attribute of the *Edit* control.

CENTER
: Centers text in an edit control. (In Windows 95 and NT this style only effects multi-line edit controls.)

RIGHT
: Aligns text flush right in an edit control. (In Windows 95 and NT this style only effects multi-line edit controls.)

UPPER
: Converts all characters to uppercase as they are typed into the edit control.

LOWER
: Converts all characters to lowercase as they are typed into the edit control.

NUMBER
: Only allows digits to be typed into the edit control. If the user types an non-digit character an information balloon is displayed informing the user that only digits are allowed. This style does not prevent the user from pasting non-digits into the control.

OEM
: Converts text entered in the edit control from the Windows character set to the OEM character set and then back again to the Windows character set. This ensures proper character conversion in certain circumstances. Microsoft states that this style is most useful when edit controls contain file names that will be used with file systems that do not support Unicode. For further details see the Windows *documentation* provided by Microsoft.

HIDDEN
: Create the edit control without the *visible* style.

DISABLED
: Create the edit control with the *disabled* style.

GROUP
: Create the edit control with the *group* style.

NOTAB
: Create the edit control without the *tabstop* style. By default the edit control is created with the tabstop style.

NOBORDER
: Create the edit control without the *border* style. By default edit controls are created with a border.

idstatic [optional]
: This argument is used to set the *resource ID* for the static control, if one is desired. If omitted, the default static control ID (-1) is used.

attr [optional]
: The name of the data *attribute* associated with the edit control. If you omit this argument then the name of the data attribute is constructed *automatically* using the text of the label. If *automatic* data detection is off, this argument is ignored.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

1
: The control was added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

Example:

The following example creates an edit control with the label *Your e-mail address* The label is placed on the edit control's left side. It also automatically creates a data *attribute* with the name YOUREMAILADDRESS. The height of the individual controls is *calculated*.

```
self~createEditInput(402, 20, 30, , 150, , "Your eMail address")
```

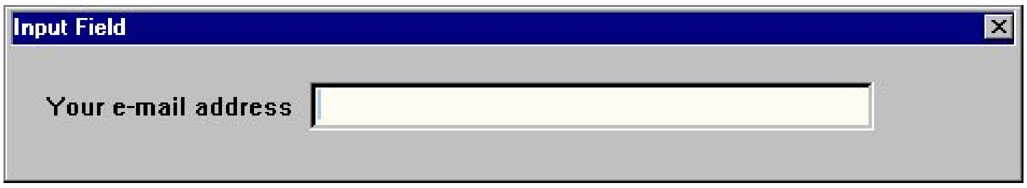


Figure 8.3. Sample Input Field

8.6.6.3. createEditInputGroup

```
                                                        +---+
                                                        V   |
>>--createEditInputGroup(-id-,-x-,-y-+-------+-,-cx2-,--txt-+-+------+-+-------+-)-><
                                     +-,-cx1-+                +-,-s-+ +-,-id2-+
```

The *createEditInputGroup* method creates a group of one or more edit controls and their static text labels in the dialog template. The height for each edit control and its label can be calculated automatically. The number of labels in the *txt* argument determines how many groups are created. A group *GroupBox* is optionally created and placed around the group if controls.

Arguments:

The arguments are:

id [required]
: The *resource ID* that is assigned to the first edit control. Consecutive numbers are assigned to the other edit controls

x, y [required]
: The position *coordinates* of the input group.

cx1 [optional]

The width coordinate of the label(s) for the edit control(s). If omitted, the width is calculated based on the dialog font.

cx2 [required]

The width coordinate for the edit control(s).

txt [required]

A single string that specifies the text used for each edit control's label. The string must consist of single words separated by blank spaces. Each word is used as the label for an edit control. Thus, the number of words determines the total number of edit controls.

If you want to use labels that include blanks (for example, "First Name" instead of "FirstName"), use the *createEditInputStem* method.

s [optional]

A list of 0 or more of the following *style* keywords separated by spaces. Each individual edit control is created with the style specified.

Note: The behavior described here for the style keywords may vary slightly depending on the version of Windows.

AUTOSCROLLH	READONLY	NUMBER
HSCROLL	KEEPSELECTION	OEM
AUTOSCROLLV	PASSWORD	HIDDEN
VSCROLL	CENTER	DISABLED
MULTILINE	RIGHT	GROUP
HIDESELECTION	UPPER	NOTAB
NOWANTRETURN	LOWER	NOBORDER

AUTOSCROLLH

Automatically scrolls text to the right when the user types a character at the end of the line. In a single-line edit control text can not be added past the width of the control without this style. Multi-line edit controls do not need this style if a horizontal scroll bar is added.

HSCROLL

Adds a horizontal scroll bar to the bottom of the edit control. A horizontal scroll bar is only functional in multi-line edit controls.

AUTOSCROLLV

Automatically scrolls text up when the user presses ENTER on the last line. This style only effects multi-line edit controls and is not needed if a vertical scroll bar is added.

VSCROLL

Adds a vertical scroll bar to the right of the edit control.

MULTILINE

Designates a multiple-line edit control. (The default is single line.) Multi-line edit controls have the WANTRETURN and KEEPSELECTION styles unless the NOWANTRETURN and / or HIDESELECTION keywords are specified.

When a multi-line edit control has the WANTRETURN style, pressing the ENTER key inserts a line break in the text of the control. Without the WANTRETURN style, pressing the ENTER key clicks the default button. With the KEEPSELECTION style, any selected text in the edit control keeps its highlight when the control loses focus.

HIDESELECTION
: Removes the KEEPSELECTION style from multi-line edit controls. This keyword only effects multi-line edit controls.

NOWANTRETURN
: Removes the WANTRETURN style from multi-line edit controls. This keyword only effects multi-line edit controls.

READONLY
: Prevents the user from entering or editing text in the edit control.

KEEPSELECTION
: Selected text in an edit control keeps its highlight when the focus is shifted away from the control.

PASSWORD
: Characters typed into the edit control are masked. Rather than displaying the character typed, the edit control displays the mask character. Prior to Windows XP the default mask character was the asterisk. In Windows XP the default mask character is a black circle. The default password character can be changed through the *passwordChar* attribute of the *Edit* control.

CENTER
: Centers text in an edit control. (In Windows 95 and NT this style only effects multi-line edit controls.)

RIGHT
: Aligns text flush right in an edit control. (In Windows 95 and NT this style only effects multi-line edit controls.)

UPPER
: Converts all characters to uppercase as they are typed into the edit control.

LOWER
: Converts all characters to lowercase as they are typed into the edit control.

NUMBER
: Only allows digits to be typed into the edit control. If the user types an non-digit character an information balloon is displayed informing the user that only digits are allowed. This style does not prevent the user from pasting non-digits into the control.

OEM
: Converts text entered in the edit control from the Windows character set to the OEM character set and then back again to the Windows character set. This ensures proper character conversion in certain circumstances. Microsoft states that this style is most useful when edit controls contain file names that will be used with file systems that do not support Unicode. For further details see the Windows *documentation* provided by Microsoft.

HIDDEN
: Create the edit control without the *visible* style.

DISABLED
: Create the edit control with the *disabled* style.

GROUP
: Create the edit control with the *group* style.

NOTAB
: Create the edit control without the *tabstop* style. By default the edit control is created with the tabstop style.

NOBORDER
: The NOBORDER style has a special meaning for this method. It controls whether or not a group box is created in the dialog template. When created the group box is used to surround the input group. If NOBORDER is used, the group box is not created, otherwise it is created.

id2 [optional]
: This argument is used to set the *resource ID* for the first static text control, that is, the label for the first edit control. If this argument is omitted, the static text controls are assigned the default (-1) static control ID.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

1
: The edit control(s) were added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

Remarks:

Note that this method calls *createEditInput* to add each individual edit control and its static text label. Because of that, a data *attribute* will be created for each edit control as though you had invoked the *createEditInput* method and omitted the *attr* argument. When the *attr* argument is omitted, the data attribute name is taken from the text of the label.

Example:

The following example creates an input group with four edit controls The single edit controls are are assigned resource IDs 301 through 304.

```
self~createEditInputGroup(301, 20, 20, , 130, "Name FirstName Street City")
```

8.6.6.4. createEditInputStem

```
>>--createEditInputStem(-id-,-x-,-y-+-------+-,-cx2-,-text.-------------------->
                                    +-,-cx1-+

>--+---------+-+----------+-)--------------------------------------------------><
   +-,-style-+ +-,-idStat-+
```

The *createEditInputStem* method creates a group of edit controls with static text labels in the dialog template. The difference between this method and the *createEditInputGroup* method is that the text for the static text controls, the labels, is passed to the method in a stem variable. This makes it possible to use labels containing blank spaces.

Arguments:

The arguments are:

id [required]

The *resource ID* that is assigned to the first edit control. Consecutive numbers are assigned to the other edit controls

x, y [required]

The position *coordinates* of the input group.

cx1 [optional]

The width coordinate for the static text control(s). If omitted, the width is calculated based on the dialog font.

cx2 [required]

The width coordinate of the edit control(s).

text. [required]

A stem containing the text used for each edit control's label. The text for the first label is put at `txt.1`, the text for the second label at `txt.2`, and so on. The number of consecutive numeric tails of the stem determines the total number of edit controls.

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces. This specifies the style for the edit control(s) in the group. Each edit control is assigned the same style.

Note: The behavior described here for the style keywords may vary slightly depending on the version of Windows.

AUTOSCROLLH	READONLY	NUMBER
HSCROLL	KEEPSELECTION	OEM
AUTOSCROLLV	PASSWORD	HIDDEN
VSCROLL	CENTER	DISABLED
MULTILINE	RIGHT	GROUP
HIDESELECTION	UPPER	NOTAB
NOWANTRETURN	LOWER	NOBORDER

AUTOSCROLLH

Automatically scrolls text to the right when the user types a character at the end of the line. In a single-line edit control text can not be added past the width of the control without this style. Multi-line edit controls do not need this style if a horizontal scroll bar is added.

HSCROLL

Adds a horizontal scroll bar to the bottom of the edit control. A horizontal scroll bar is only functional in multi-line edit controls.

AUTOSCROLLV

Automatically scrolls text up when the user presses ENTER on the last line. This style only effects multi-line edit controls and is not needed if a vertical scroll bar is added.

VSCROLL
: Adds a vertical scroll bar to the right of the edit control.

MULTILINE
: Designates a multiple-line edit control. (The default is single line.) Multi-line edit controls have the WANTRETURN and KEEPSELECTION styles unless the NOWANTRETURN and / or HIDESELECTION keywords are specified.

  When a multi-line edit control has the WANTRETURN style, pressing the ENTER key inserts a line break in the text of the control. Without the WANTRETURN style, pressing the ENTER key clicks the default button. With the KEEPSELECTION style, any selected text in the edit control keeps its highlight when the control loses focus.

HIDESELECTION
: Removes the KEEPSELECTION style from multi-line edit controls. This keyword only effects multi-line edit controls.

NOWANTRETURN
: Removes the WANTRETURN style from multi-line edit controls. This keyword only effects multi-line edit controls.

READONLY
: Prevents the user from entering or editing text in the edit control.

KEEPSELECTION
: Selected text in an edit control keeps its highlight when the focus is shifted away from the control.

PASSWORD
: Characters typed into the edit control are masked. Rather than displaying the character typed, the edit control displays the mask character. Prior to Windows XP the default mask character was the asterisk. In Windows XP the default mask character is a black circle. The default password character can be changed through the *passwordChar* attribute of the *Edit* control.

CENTER
: Centers text in an edit control. (In Windows 95 and NT this style only effects multi-line edit controls.)

RIGHT
: Aligns text flush right in an edit control. (In Windows 95 and NT this style only effects multi-line edit controls.)

UPPER
: Converts all characters to uppercase as they are typed into the edit control.

LOWER
: Converts all characters to lowercase as they are typed into the edit control.

NUMBER
: Only allows digits to be typed into the edit control. If the user types an non-digit character an information balloon is displayed informing the user that only digits are allowed. This style does not prevent the user from pasting non-digits into the control.

OEM

Converts text entered in the edit control from the Windows character set to the OEM character set and then back again to the Windows character set. This ensures proper character conversion in certain circumstances. Microsoft states that this style is most useful when edit controls contain file names that will be used with file systems that do not support Unicode. For further details see the Windows *documentation* provided by Microsoft.

HIDDEN

Create the edit control without the *visible* style.

DISABLED

Create the edit control with the *disabled* style.

GROUP

Create the edit control with the *group* style.

NOTAB

Create the edit control without the *tabstop* style. By default the edit control is created with the tabstop style.

NOBORDER

The NOBORDER style has a special meaning for this method. It controls whether or not a group box is created in the dialog template. When created, the group box is set to surround the input group. If NOBORDER is used, the group box is not added to the dialog template, otherwise it is added to the dialog template.

idStat [optional]

This argument is used to set the *resource ID* for the first static text control, that is, the label for the first edit control. If this argument is omitted the static text controls are assigned the default (-1) static control ID. Resource IDs only need to be assigned to the static controls if there is some need to manipulate the underlying static control after it has been created. For instance if its text needed to be changed, or its position needed to be changed.

Return value:

The possible return values are:

-2

The dialog template has either not been started or is complete.

-1

A symbolic ID was used, but it could not be resolved correctly.

0

Success.

1

The edit control(s) were added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

Remarks:

Note that this method calls *createEditInput* to add each individual edit control and its static text label. Because of that, a data *attribute* will be created for each edit control as though you had invoked the *createEditInput* method and omitted the *attr* argument. When the *attr* argument is omitted, the data attribute name is taken from the text of the label.

Example:

The following example shows how to use *createEditInputStem*. It creates four edit controls with static text labels. For each edit control (with IDs 401 through 404) the text for the label, (the static text control,) is provided by the stem variable.

The labels also provide the name attribute name for each edit control. The attribute names will be slightly different from the title because the spaces and ampersands are removed. In this example the NAME, FIRSTNAME, STREETNUMBER, and CITYZIP attributes are added to the object.

```
FNames.1="Name"
FNames.2="First Name"
FNames.3="Street & Number"
FNames.4="City & ZIP"

self~createEditInputStem(401, 20, 20, , 150, FNames.)
```

8.6.6.5. createPasswordEdit

```
>>--createPasswordEdit(-id-,-x-,-y-,-cx--+------+--+---------+-+--------+--)---><
                                         +-,-cy-+  +-,-style-+ +-,-attr-+
```

The *createPasswordEdit* method creates an edit edit control in the dialog template with the PASSWORD style.

When an edit control has the *PASSWORD* style it does not echo the characters typed but displays the password character instead. This method is a convenience method and is used exactly like the *createEdit* method. It simple specifies the PASSWORD style automatically for the programmer.

Arguments:

The arguments are:

id [required]

The *resource ID* for the control.

x, y, cx [required]

The position and width *coordinates* for the control.

cy [optional]

The height *coordinates* for the control. If this argument is omitted, then the height is calculated to fit the height of the dialog font.

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces.

Note: The behavior described here for the style keywords may vary slightly depending on the version of Windows.

AUTOSCROLLH	READONLY	OEM
HSCROLL	KEEPSELECTION	HIDDEN
AUTOSCROLLV	CENTER	DISABLED
VSCROLL	RIGHT	GROUP
MULTILINE	UPPER	NOTAB
HIDESELECTION	LOWER	NOBORDER
NOWANTRETURN	NUMBER	

AUTOSCROLLH
: Automatically scrolls text to the right when the user types a character at the end of the line. In a single-line edit control text can not be added past the width of the control without this style. Multi-line edit controls do not need this style if a horizontal scroll bar is added.

HSCROLL
: Adds a horizontal scroll bar to the bottom of the edit control. A horizontal scroll bar is only functional in multi-line edit controls.

AUTOSCROLLV
: Automatically scrolls text up when the user presses ENTER on the last line. This style only effects multi-line edit controls and is not needed if a vertical scroll bar is added.

VSCROLL
: Adds a vertical scroll bar to the right of the edit control.

MULTILINE
: Designates a multiple-line edit control. (The default is single line.) Multi-line edit controls have the WANTRETURN and KEEPSELECTION styles unless the NOWANTRETURN and / or HIDESELECTION keywords are specified.

: When a multi-line edit control has the WANTRETURN style, pressing the ENTER key inserts a line break in the text of the control. Without the WANTRETURN style, pressing the ENTER key clicks the default button. With the KEEPSELECTION style, any selected text in the edit control keeps its highlight when the control loses focus.

HIDESELECTION
: Removes the KEEPSELECTION style from multi-line edit controls. This keyword only effects multi-line edit controls.

NOWANTRETURN
: Removes the WANTRETURN style from multi-line edit controls. This keyword only effects multi-line edit controls.

READONLY
: Prevents the user from entering or editing text in the edit control.

KEEPSELECTION
: Selected text in an edit control keeps its highlight when the focus is shifted away from the control.

CENTER
: Centers text in an edit control. (In Windows 95 and NT this style only effects multi-line edit controls.)

RIGHT
: Aligns text flush right in an edit control. (In Windows 95 and NT this style only effects multi-line edit controls.)

UPPER
: Converts all characters to uppercase as they are typed into the edit control.

LOWER
: Converts all characters to lowercase as they are typed into the edit control.

NUMBER

Only allows digits to be typed into the edit control. If the user types an non-digit character an information balloon is displayed informing the user that only digits are allowed. This style does not prevent the user from pasting non-digits into the control.

OEM

Converts text entered in the edit control from the Windows character set to the OEM character set and then back again to the Windows character set. This ensures proper character conversion in certain circumstances. Microsoft states that this style is most useful when edit controls contain file names that will be used with file systems that do not support Unicode. For further details see the Windows *documentation* provided by Microsoft.

HIDDEN

Create the edit control without the *visible* style.

DISABLED

Create the edit control with the *disabled* style.

GROUP

Create the edit control with the *group* style.

NOTAB

Create the edit control without the *tabstop* style. By default the edit control is created with the tabstop style.

NOBORDER

Create the edit control without the *border* style. By default edit controls are created with a border.

attr [optional]

The name of the data *attribute* associated with the dialog control. If you omit this argument then the name of the data attribute is constructed *automatically*. If *automatic* data detection is off, this argument is ignored.

Return value:

The possible return values are:

-2

The dialog template has either not been started or is complete.

-1

A symbolic ID was used, but it could not be resolved correctly.

0

Success.

1

The control was added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

8.6.7. createListBox

```
>>--createListBox(--id-,-x-,-y-,-cx-,-cy--+---------+-+-----------+-)---------><
```

```
                                                  +-,-style-+ +-,-attrName-+
```

Adds a list *ListBox* to the dialog template.

Arguments:

The arguments are:

id [required]

The *resource ID* for the control.

x, y, cx, cy [required]

The control *coordinates*.

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces:

MULTI	MCOLUMN	GROUP
NOTIFY	PARTIAL	DISABLED
SORT	SBALWAYS	NOBORDER
COLUMNS	KEYINPUT	NOTAB
VSCROLL	EXTSEL	
HSCROLL	HIDDEN	

MULTI

Makes the list box a multiple choice list box, that is, you can select more than one line.

NOTIFY

Informs the list box that it should send the DBLCLICK, SELCHANGE, and SELCANCEL *event* notifications. The list box sends these event notifications only when it has the NOTIFY style. The other list box event notifications are always sent. Use the *connectListBoxEvent* method to connect event notifications to a method in the Rexx dialog.

SORT

The items in the dialog are listed in the noted order.

COLUMNS

The list box can handle tab characters ("09"x). Use this option together with the *setListTabulators* method to have more than one column in a list.

VSCROLL

Adds a vertical scroll bar to the list box. Scroll bars appear only if the list contains more lines than can fit in the available space.

HSCROLL

Adds a horizontal scroll bar to the list box. See also *setListWidth*.

MCOLUMN

Makes the list box a multicolumn list box that can be scrolled horizontally. *setListColumnWidth* sets the width of the columns.

PARTIAL

The size of the list box equals the size specified by the application when it created the list box. Windows usually sizes a list box such that the list box does not display partial items.

SBALWAYS

The list box shows a disabled scroll bar if there is no need to scroll. If you do not specify this option, the scroll bar is hidden when the list box does not contain enough items.

KEYINPUT

Specifies that the owner of the list box receives WM_VKEYTOITEM messages whenever the user presses a key and the list box has the input focus. This enables an application to perform special processing on the keyboard input.

To take advantage of this style, the programmer would need to use the *addUserMsg*() method to receive the keyboard event notification. Future versions of ooDialog may support this event directly.

EXTSEL

Allows multiple items to be selected by using the SHIFT key and the mouse or special key combinations.

HIDDEN

Create the list box without the *visible* style.

DISABLED

Create the list box with the *disabled* style.

GROUP

Create the list box with the *group* style.

NOTAB

Create the list box without the *tabstop* style. By default the list box is created with the tabstop style.

NOBORDER

Create the list box without the *border* style. By default list box controls are created with a border.

attrName [optional]

The name of the data *attribute* associated with the list box control. If you omit this argument then the name of the data attribute is constructed *automatically*. If *automatic* data detection is off, this argument is ignored.

Return value:

The possible return values are:

-2

The dialog template has either not been started or is complete.

-1

A symbolic ID was used, but it could not be resolved correctly.

0

Success.

1

The control was added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

8.6.8. createListView

```
>>--createListView(-id-,--x-,--y-,--cx-,--cy-+---------+--+------------+--)----><
                                             +-,-style-+  +-,-attrName-+
```

Creates a *ListView* control in the dialog template and specifies the resource ID, position, size, style, and attribute name for the control.

Arguments:

The arguments are:

id [required]

The *resource ID* for the list-view control. May be numeric or *symbolic*.

x, y, cx, cy [required]

The control *coordinates*

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces:

ICON	DESCENDING	SINGLESEL
SMALLICON	EDIT	SHOWSELALWAYS
LIST	HSCROLL	SHAREIMAGES
REPORT	VSCROLL	HIDDEN
ALIGNLEFT	NOSCROLL	DISABLED
ALIGNTOP	NOHEADER	GROUP
AUTOARRANGE	NOSORTHEADER	BORDER
ASCENDING	NOWRAP	TAB

ICON

Use the icon view. Each item appears as a full-sized icon with a label below it. The user can drag the items to any location in the list-view control.

SMALLICON

Use the small-icon view. Each item appears as a small icon with a label to the right of it. The user can drag the items to any location in the list-view control.

LIST

Use the list view. Each item appears as a small icon with a label to the right of it. Items are arranged in columns and cannot be moved by the user.

REPORT

Use the report view. Each item appears on a separate line with information arranged in columns. The leftmost column contains the small icon and label, and subsequent columns contain subitems.

ALIGNLEFT

In icon and small-icon views, the items are left-aligned.

ALIGNTOP

In icon and small-icon views, the items are aligned with the top of the control.

AUTOARRANGE

In icon and small-icon views, the icons are always automatically arranged.

ASCENDING
: Sorts items by item text in ascending order.

DESCENDING
: Sorts items by item text in descending order.

EDIT
: With this style, the list-view allows the user to edit the list-view item labels. To have an edited label take effect, the programmer must connect an *event* notification using the *connectListViewEvent* method. For the most flexibility with edit labeling, connect both the *BEGINEDIT* and *ENDEDIT* notifications. Alternatively the *DEFAULTEDIT* event could be connected using the *connectListViewEvent* method.

HSCROLL
: The list-view supports a horizontal scroll bar.

VSCROLL
: The list-view supports a vertical scroll bar.

NOSCROLL
: Disables scrolling.

NOHEADER
: No column header is displayed in the report view. By default, columns have headers in the report view.

NOSORTHEADER
: Specifies that column headers do not work like buttons. This option is useful if clicking a header in the report view does not carry out an action.

NOWRAP
: Displays item text on a single line in the icon view. By default, the item text can wrap in the icon view.

SINGLESEL
: Allows only one item to be selected at a time. By default, several items can be selected.

SHOWSELALWAYS
: Specifies that a selected item remains selected when the list-view loses focus.

SHAREIMAGES
: The list-view does not take ownership of the image lists assigned to it. This option enables an *ImageList* list to be used with several list controls.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

GROUP
: Create the list-view with the *group* style.

NOBORDER
: Create the list-view without *border* window style. By default list views are created with a border

NOTAB
: Create the list-view without the *tabstop* control style. By default the control is created with the tabstop style.

attributeName [optional]
: The name of the data *attribute* associated with the dialog control. If you omit this argument then the name of the data attribute is constructed *automatically*. If *automatic* data detection is off, this argument is ignored.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

1
: The control was added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

Example:

The following example creates a list-view with the resource ID of 555 at position x=100 and y=80 and with a size of width=40 and height=120. The list-view is a report view with items sorted in ascending order. It supports item editing and column headers do not behave like buttons. The data *attribute* for the list-view is EMPLOYEES.

```
self~createListView(555, 100, 80, 40, 120, "REPORT ASCENDING EDIT NOSORTHEADER",
 "EMPLOYEES")
```

8.6.9. createMonthCalendar

```
>>--createMonthCalendar(-id--x-,--y-,--cx-,--cy-+---------+--+------------+--)-><
                                                +-,-style-+  +-,-attrName-+
```

Creates a month *MonthCalendar* control in the dialog template.

Arguments:

The arguments are:

id [required]
: The *resource ID* for the month calendar. May be numeric or *symbolic*.

x, y, cx, cy [required]
: The control *coordinates*

style [optional]
: A list of 0 or more of the following *style* keywords separated by spaces:

DAYSTATE	NOTRAILING	GROUP
MULTI	SHORTDAYS	BORDER

NOTODAY	NOSELCHANGE	NOTAB
NOCIRCLE	HIDDEN	
WEEKNUMBERS	DISABLED	

DAYSTATE

The month calendar will send the GETDAYSTATE *event* notification to request information about which days should be displayed in bold. Use the *connectMonthCalendarEvent* method to connect this notification with a method in the Rexx dialog object.

MULTI

This style allows the user to select a range of dates within the control. By default, the maximum range is one week. The programmer changes the maximum range that can be selected by using the *setRange* method.

NOTODAY

The month calendar control will not display the *today* date at the bottom of the control.

NOCIRCLE

The month calendar control will not circle the *today* date at the bottom of the control.

WEEKNUMBERS

The month calendar displays week numbers (1-52) to the left of each row of days. Week 1 is defined by Microsoft as the first week that contains at least four days.

NOTRAILING

Requires Windows Vista or later. This style disables displaying the dates from the previous / next month in the current calendar.

SHORTDAYS

Requires Windows Vista or later. With this style, the month calendar uses the shortest day name for the label of the day of the week column header.

NOSELCHANGE

Requires Windows Vista or later. With this style the selection does not change when the user navigates to he next or previous month in the calendar. This style is needed for the user to be able to select a range greater than what is currently displayed.

HIDDEN

The not *visible* window style.

DISABLED

The *disabled* window style.

BORDER

The *border* window style.

GROUP

The *group* control style.

NOTAB

The no *tabstop* control style.

attrName [optional]

The name of the data *attribute* associated with the dialog control. If you omit this argument then the name of the data attribute is constructed *automatically*. If *automatic* data detection is off, this argument is ignored.

Return value:

The possible return values are:

-2

The dialog template has either not been started or is complete.

-1

A symbolic ID was used, but it could not be resolved correctly.

0

Success.

1

The control was added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

Example:

The following example creates a month calendar control with the *symbolic* ID of IDC_MC1 positioned at (162, 7) of the dialog's *client area*. The control will be 140 *dialog unit* wide and 100 dialog units high. The calendar will not display the *today* date at the bottom of the calendar and will not circle the *today* date. The DAYSTATE style indicates that the programmer wants to receive the *GETDAYSTATE* event notifications.

```
self~createMonthCalendar(IDC_MC1, 162, 7, 140, 100, "NOTODAY NOCIRCLE DAYSTATE")
```

8.6.10. createProgressBar

```
>>--createProgressBar(--id--,--x--,--y--,--cx--,--cy-+-----------+--)----------><
                                                     +--,-style--+
```

The *createProgressBar* method creates a progress *ProgressBar* in the dialog template and connects it with a data *attribute*.

Arguments:

The arguments are:

id [required]

The *resource ID* for the progress bar.

x, y, cx, cy [required]

The control *coordinates*.

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces.

VERTICAL	HIDDEN	GROUP
SMOOTH	DISABLED	TAB
MARQUEE	BORDER	

VERTICAL

The progress bar is orientated vertically. The default orientation is horizontal.

SMOOTH
: The progress is displayed in a smooth scrolling bar rather than the default segmented bar. This style is only displayed when Windows classic theme is in effect. All other themes over-ride this style.

MARQUEE
: The progress bar moves like a marquee. This method requires Common Control *Library* version 6.0 or later.

HIDDEN
: The progress bar has the not *visible* window style.

DISABLED
: The progress bar has the *disabled* window style.

BORDER
: The progress bar has the *border* window style. Normally the progress bar is created without the border style.

GROUP
: The progress bar has the *group* control style. A progress bar is normally created without the group style.

TAB
: The progress bar has the *tabstop* control style. Normally, a progress bar is not a tab stop.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

Example:

The following example creates a progress bar with the *symbolic* resource id, IDC_PBAR_DONE, close to the bottom of the dialog. The progress bar is 12 dialog units high with a border of 10 dialog units below it. The progress bar is almost as wide as the dialog. It has a border of 10 dialog units on each side.

```
self~createProgressBar("IDC_PBAR_DONE", 10, self~sizeY - 22, self~sizeX - 20, 12)
```

8.6.11. createReBar

```
>>--createReBar(-id-,--x-,--y-,--cx-,--cy-+---------+--+----------------+--)-><
                                          +-,-style-+  +-,-attributeName-+
```

Creates a *ReBar* control in the dialog template and specifies the resource ID, position, size, and style for the control.

Arguments:

The arguments are:

id [required]

The *resource ID* for the rebar control. May be numeric or *symbolic*.

x, y, cx, cy [required]

The control *coordinates*. Rebar controls typically ignore the coordinates, they position and size themselves according to their internal logic.

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces. Case is not significant:

AUTOSIZE	TOOLTIPS	GROUP
BANDBORDERS	VARHEIGHT	HIDDEN
DBLCLKTOGGLE	VERTICALGRIPPER	TAB
FIXEDORDER	BORDER	
REGISTERDROP	DISABLED	

AUTOSIZE

The rebar control will automatically change the layout of the bands when the size or position of the control changes. An AUTOSIZE notification will be sent when this occurs.

BANDBORDERS

The rebar control displays narrow lines to separate adjacent bands.

DBLCLKTOGGLE

The rebar band will toggle its maximized or minimized state when the user double-clicks the band. Without this style, the maximized or minimized state is toggled when the user single-clicks on the band.

FIXEDORDER

The rebar control always displays bands in the same order. You can move bands to different rows, but the band order is static.

REGISTERDROP

The rebar control generates GETOBJECT notification messages when an object is dragged over a band in the control. Although the ooDialog framework supports this style and event notification, there is no way within the framework to make use of it.

TOOLTIPS

This style keyword is for completeness, the rebar control itself does not yet support tool tips.

VARHEIGHT

The rebar control displays bands at the minimum required height, when possible. Without this style, the rebar control displays all bands at the same height, using the height of the tallest visible band to determine the height of other bands.

VERTICALGRIPPER

The size grip will be displayed vertically instead of horizontally in a vertical rebar control. This style is ignored for rebar controls that do not have the CCS_VERT style.

BORDER

Adds the *border* window style. By default the BORDER style is not added. However, rebar controls seem to ignore this style completely. The *createRebar* method accepts this keyword so that the ooDialog programmer can experiment with it. But it is not expected that using BORDER will produce any noticeable difference in appearance.

DISABLED
: Gives the rebar control the *disabled* window style. By default the control is enabled

GROUP
: The *group* control style is added to the control. By default the GROUP style is not used.

HIDDEN
: The not *visible* window style.

TAB
: Create the control with the *tabstop* control style. By default the control is created without the TABSTOP style. Note that the behavior of the rebar control does not change with, or without, the TABSTOP style. The *createRebar* method accepts this keyword so that the ooDialog programmer can experiment with it. But it is not expected that using TAB will produce any noticeable difference in behavior.

In addition, the *createReBar* method has support for any *Common Control Styles* keyword.

attributeName [optional]
: Although this argument is accepted so that the format of the *createToolBar* method is similar to the other *createXX* methods, the concept of a rebar having *data* makes no sense. If the argument is used it is ignored.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

Remarks:

Although Microsoft's documentation implies that the *Common Control Styles* are ignored by the rebar control, experimentation has shown that the control does not ignore them.

A rebar window can also be dynamically added to an **`UserDialog`** through the *createReBarWindow* method

Details

Raises syntax errors when incorrect usage is detected.

Example:

The following example creates a rebar control that displays the bands at the minimum required height when possible, has narrow lines between adjacent bands, and keeps the bands in the same order:

```
self~createReBar(IDC_RBAR, 0, 1, 220, 22, "VARHEIGHT BANDBORDERS FIXEDORDER")
```

8.6.12. createScrollBar

```
>>--createScrollBar(--id--,--x--,--y--,--cx--,--cy--+----------+--)------------><
                                                    +-,-style--+
```

The *createScrollBar* method creates a scroll bar in the dialog template.

Arguments:

The arguments are:

id [required]

The *resource ID* for the scroll bar

x, y, cx, cy [required]

The control *coordinates*.

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces:

VERTICAL	BOTTOMRIGHT	DISABLED
HORIZONTAL	HIDDEN	BORDER
TOPLEFT	GROUP	TAB

VERTICAL

The scroll bar is positioned vertically. This is the default if HORIZONTAL is not specified.

HORIZONTAL

The scroll bar is positioned horizontally.

TOPLEFT

The scroll bar is aligned to the top left of the given rectangle and has a predetermined width (if vertical) or height (if horizontal.)

BOTTOMRIGHT

The scroll bar is aligned to the bottom right of the given rectangle and has a predetermined width (if vertical) or height (if horizontal.)

HIDDEN

Create the scroll bar without the *visible* style.

DISABLED

Create the scroll bar with the *disabled* style.

GROUP

Create the scroll bar with the *group* style.

TAB

Create the scroll bar with the *tabstop* style.

BORDER

Create the scroll bar with the *border* style.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

8.6.13. Create Static Controls

The methods in this section are all used to create a static control in the dialog template. However, there is no need to use a number of different methods to create static controls. The *createStatic* method is all that is needed. You control the appearance and behavior of the static control in the same way as with other controls, by specifying the appropriate control styles. Given the proper combination of style keywords, the **`createStatic`** method can create any static control that the other methods in this section create.

However, there are a large number of style keywords for the static control. Many of the keywords have no meaning when used in combination with other keywords. For instance, the ENDELLIPSIS keyword has no meaning if used with the ICON keyword. The other methods in this section are convenience methods where the number of style keywords are reduced to only those that make sense in combination.

A word about frames and rectangles. In early versions of the ooDialog reference frames and rectangles were documented as though they were types of separate controls. They are not, they are just a static control with a particular style. The Frames and Rectangles picture gives some idea of how the different types of static controls can appear:

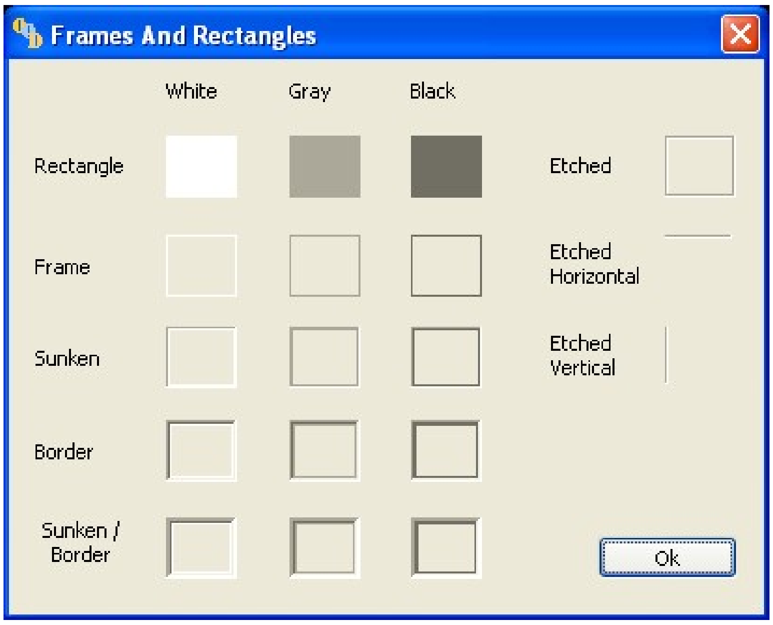


Figure 8.4. Frames and Rectangles

8.6.13.1. createBlackFrame

```
>>--createBlackFrame(--+------+--x-,-y-,-cx-,-cy--+-----------+-)--------------><
```

```
                         +--id--+                    +-,--style--+
```

The *createBlackFrame* method creates a black *frame* static control in the dialog template. The rectangle is drawn with the current window frame color, which is black in the default color scheme.

Arguments:

The arguments are:

id [optional]

The *resource ID* for the static control.

x, y, cx, cy [required]

The control *coordinates*

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces.

NOTIFY	DISABLED	TAB
SUNKEN	GROUP	
HIDDEN	BORDER	

NOTIFY

Causes the static control to send notification messages for the click, double click, disabled, and enabled events. A static control without this style does not send notification messages. See the *connectStaticEvent* method of the *EventNotification* class for further information.

SUNKEN

The control is drawn with a half-sunken border around it.

HIDDEN

The not *visible* window style.

DISABLED

The *disabled* window style.

BORDER

The *border* window style.

GROUP

The *group* control style.

TAB

The *tabstop* control style.

Return value:

The possible return values are:

-2

The dialog template has either not been started or is complete.

-1

A symbolic ID was used, but it could not be resolved correctly.

0

Success.

Example:

See the Frames and Rectangles *example* .

8.6.13.2. createBlackRect

```
>>--createBlackRect(--+------+--x-,-y-,-cx-,-cy--+----------+-)--------------><
                      +--id--+                   +-,--style--+
```

The *createBlackRect* method creates a black *rectangle*) static control in the dialog template. The rectangle is filled with the current window frame color, which is black in the default color scheme.

Arguments:

The arguments are:

id [optional]

The *resource ID* for the static control.

x, y, cx, cy [required]

The control *coordinates*

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces.

NOTIFY	DISABLED	TAB
SUNKEN	GROUP	
HIDDEN	BORDER	

NOTIFY

Causes the static control to send notification messages for the click, double click, disabled, and enabled events. A static control without this style does not send notification messages. See the *connectStaticEvent* method of the *EventNotification* class for further information.

SUNKEN

The control is drawn with a half-sunken border around it.

HIDDEN

The not *visible* window style.

DISABLED

The *disabled* window style.

BORDER

The *border* window style.

GROUP

The *group* control style.

TAB

The *tabstop* control style.

Return value:

The possible return values are:

-2

The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

Example:

See the Frames and Rectangles *example* .

8.6.13.3. createEtchedFrame

```
>>--createEtchedFrame(--+-------+--x-,-y-,-cx-,-cy--+-----------+-)------------><
                        +--id-,-+                   +-,--style--+
```

The *createEtchedFrame*() method adds an etched *frame* static control to the dialog.

Arguments:

The arguments are:

id [optional]
: The *resource ID* for the static control.

x, y, cx, cy [required]
: The control *coordinates*

style [optional]
: A list of 0 or more of the following *style* keywords separated by spaces.

NOTIFY	DISABLED	TAB
SUNKEN	GROUP	
HIDDEN	BORDER	

NOTIFY
: Causes the static control to send notification messages for the click, double click, disabled, and enabled events. A static control without this style does not send notification messages. See the *connectStaticEvent* method of the *EventNotification* class for further information.

SUNKEN
: The control is drawn with a half-sunken border around it.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

BORDER
: The *border* window style.

GROUP
: The *group* control style.

TAB
: The *tabstop* control style.

Return value:

The possible return values are:

-2

The dialog template has either not been started or is complete.

-1

A symbolic ID was used, but it could not be resolved correctly.

0

Success.

Example:

See the Frames and Rectangles *example* .

8.6.13.4. createEtchedHorizontal

```
>>--createEtchedHorizontal(--+-------+--x-,-y-,-cx-,-cy--+-----------+-)-------><
                             +--id-,-+                    +-,--style--+
```

The *createEtchedHorizontal*() method adds an etched horizontal *line* static control to the dialog.

Arguments:

The arguments are:

id [optional]

The *resource ID* for the static control.

x, y, cx, cy [required]

The control *coordinates*

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces.

NOTIFY	DISABLED	TAB
SUNKEN	GROUP	
HIDDEN	BORDER	

NOTIFY

Causes the static control to send notification messages for the click, double click, disabled, and enabled events. A static control without this style does not send notification messages. See the *connectStaticEvent* method of the *EventNotification* class for further information.

SUNKEN

The control is drawn with a half-sunken border around it.

HIDDEN

The not *visible* window style.

DISABLED

The *disabled* window style.

BORDER

The *border* window style.

GROUP
: The *group* control style.

TAB
: The *tabstop* control style.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

Example:

See the Frames and Rectangles *example* .

8.6.13.5. createEtchedVertical

```
>>--createEtchedVertical(--+-------+--x-,-y-,-cx-,-cy--+-----------+-)---------><
                           +--id-,-+                   +-,--style--+
```

The *createEtchedVertical*() method creates an etched vertical *line* static control in the dialog template.

Arguments:

The arguments are:

id [optional]
: The *resource ID* for the static control.

x, y, cx, cy [required]
: The control *coordinates*

style [optional]
: A list of 0 or more of the following *style* keywords separated by spaces.

NOTIFY	DISABLED	TAB
SUNKEN	GROUP	
HIDDEN	BORDER	

NOTIFY
: Causes the static control to send notification messages for the click, double click, disabled, and enabled events. A static control without this style does not send notification messages. See the *connectStaticEvent* method of the *EventNotification* class for further information.

SUNKEN
: The control is drawn with a half-sunken border around it.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

BORDER
: The *border* window style.

GROUP
: The *group* control style.

TAB
: The *tabstop* control style.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

Example:

See the Frames and Rectangles *example* .

8.6.13.6. createGrayFrame

```
>>--createGrayFrame(--+------+--x-,-y-,-cx-,-cy--+-----------+-)---------------><
                      +--id--+                   +-,--style--+
```

The *createGrayFrame* method creates a gray *frame* static control in the dialog template. The frame is drawn with the same color as the screen background (desktop), which is gray in the default color scheme.

Arguments:

The arguments are:

id [optional]
: The *resource ID* for the static control.

x, y, cx, cy [required]
: The control *coordinates*

style [optional]
: A list of 0 or more of the following *style* keywords separated by spaces.

NOTIFY	DISABLED	TAB
SUNKEN	GROUP	
HIDDEN	BORDER	

NOTIFY
: Causes the static control to send notification messages for the click, double click, disabled, and enabled events. A static control without this style does not send notification

messages. See the *connectStaticEvent* method of the *EventNotification* class for further information.

SUNKEN
: The control is drawn with a half-sunken border around it.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

BORDER
: The *border* window style.

GROUP
: The *group* control style.

TAB
: The *tabstop* control style.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

Example:

See the Frames and Rectangles *example* .

8.6.13.7. createGrayRect

```
>>--createGrayRect(--+------+--x-,-y-,-cx-,-cy--+-----------+-)----------------><
                     +--id--+                   +-,--style--+
```

The *createGrayRect* method creates a gray *rectangle*) static control in the dialog template. The rectangle is filled with the same color as the screen background (desktop), which is gray in the default color scheme.

Arguments:

The arguments are:

id [optional]
: The *resource ID* for the static control.

x, y, cx, cy [required]
: The control *coordinates*

style [optional]
: A list of 0 or more of the following *style* keywords separated by spaces.

NOTIFY	DISABLED	TAB
SUNKEN	GROUP	
HIDDEN	BORDER	

NOTIFY
: Causes the static control to send notification messages for the click, double click, disabled, and enabled events. A static control without this style does not send notification messages. See the *connectStaticEvent* method of the *EventNotification* class for further information.

SUNKEN
: The control is drawn with a half-sunken border around it.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

BORDER
: The *border* window style.

GROUP
: The *group* control style.

TAB
: The *tabstop* control style.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

Example:

See the Frames and Rectangles *example* .

8.6.13.8. createStatic

```
>>--createStatic(--+----+--,-x-,-y-,-cx-,-cy--+----------+--+---------+--)----><
                   +-id-+                     +-,-style--+  +-,-text--+
```

The *createStatic* method is used to create a static control in the dialog template. It is a generic method that can be used for any type of static control. The 3 basic types of static controls are static text, static image, and static frames / rectangles.

Arguments:

The arguments are:

id [optional]

The *resource ID* for the static control. -1 is used if *id* is omitted.

x, y, cx, cy [required]

The control *coordinates*

style [optional]

A list of 0 or more *style* keywords separated by spaces.

TEXT	VERT	WORDELLIPSIS
BITMAP	LEFT	CENTERIMAGE
METAFILE	CENTER	RIGHTJUST
ICON	RIGHT	SIZECONTROL
WHITERECT	SIMPLE	SIZEIMAGE
GRAYRECT	LEFTNOWRAP	HIDDEN
BLACKRECT	NOTIFY	DISABLED
WHITEFRAME	SUNKEN	GROUP
GRAYFRAME	EDITCONTROL	BORDER
BLACKFRAME	ENDELLIPSIS	TAB
ETCHED	NOPREFIX	
HORZ	PATHELLIPSIS	

The keywords: TEXT, BITMAP, METAFILE, ICON, WHITERECT, GRAYRECT, BLACKRECT, WHITEFRAME, GRAYFRAME, BLACKFRAME, ETCHED, HORZ, and VERT determine the type of static control to be created. If more than one of these type keywords is specified, which static control is created is undefined. The other style keywords modify the appearance or behavior of the static control.

TEXT

A static text control is created, this is the default if no other static type is specified.

BITMAP

A static image control is created that will use a bitmap.

METAFILE

A static image control that uses a metafile will be created.

ICON

A static image control that uses an icon will be created. Cursors are also icons.

WHITERECT

A *rectangle*)rectangle filled with the current window background color, which is white in the default color scheme.

WHITEFRAME

A *frame* drawn with the same color as the window background, which is white in the default color scheme.

GRAYRECT

A *rectangle*)rectangle filled with the current screen background color, which is gray in the default color scheme.

GRAYFRAME

A *frame* drawn with the same color as the current screen background, which is gray in the default color scheme.

BLACKRECT
: A *rectangle*)rectangle filled with the current window frame color, which is black in the default color scheme.

BLACKFRAME
: A *frame* drawn with the same color as the current window frames, which is black in the default color scheme.

ETCHED
: A *frame* drawn with an etched appearance.

HORZ
: A *frame* drawn with an etched appearance, but only the top horizontal line of the frame is drawn. In other words, this is a single horizontal line, whose position and length are determined by the position and width of the imaginary frame specified by the control coordinates.

VERT
: A *frame* drawn with an etched appearance, but only the left vertical line of the frame is drawn. In other words, this is a single vertical line, whose position and length are determined by the position and height of the imaginary frame specified by the control coordinates.

LEFT
: Left aligns text for static text controls. Lines are automatically word wrapped. Words longer than the width of the control are truncated. This is the default for static text controls and does not need to be specified.

CENTER
: Uses centered text alignment for static text controls. Lines are automatically word wrapped. Words longer than the width of the control are truncated. If no text alignment style is specified, the text is left aligned.

RIGHT
: Right aligns text for static text controls. Lines are automatically word wrapped. Words longer than the width of the control are truncated. If no text alignment style is specified, the text is left aligned.

SIMPLE
: A single line of left-aligned text is in the rectangle defined by the control coordinates. The text line cannot be shortened or altered in any way. If the control is disabled, the text is not grayed.

LEFTNOWRAP
: A single line of text, left-aligned. Tabs are expanded, but words are not wrapped. Text that extends past the end of a line is clipped.

NOTIFY
: Causes the static control to send notification messages for the click, double click, disabled, and enabled events. A static control without this style does not send notification messages. See the *connectStaticEvent* method of the *EventNotification* class for further information.

SUNKEN
: The control is drawn with a half-sunken border around it.

EDITCONTROL
: The static control will act like a multi-line edit control when it displays text. This consists of calculating the average character width in the same manner as the edit control, and not displaying a partially visible last line.

ENDELLIPSIS
: If the text does not fit in the rectangle, it is truncated and an ellipsis is added. Compare this to PATHELLIPSIS.

NOPREFIX
: Any ampersand (&) characters in the control's text are not interpreted as accelerator prefix characters. The text is displayed with the ampersand removed and the next character in the text underlined.

PATHELLIPSIS
: If the text is too big for the specified rectangle, characters in the middle of the text are replaced with an ellipsis so that the result fits in the rectangle. If the text contains backslash (\) characters, this style preserves as much as possible of the text after the last backslash.

WORDELLIPSIS
: Any word that does not fit in the rectangle is truncated and ellipses are added.

CENTERIMAGE
: Bitmaps are centered in the static control that contains it. The control is not resized, so that a bitmap too large for the control will be clipped. If the static control contains a single line of text, the text is centered vertically in the client area of the control.

RIGHTJUST
: The lower right corner of the static control with the BITMAP or ICON style remains fixed when the control is resized. Only the top and left sides are adjusted to accommodate a new bitmap or icon.

SIZECONTROL
: Adjusts the bitmap to fit the size of the static control. For example, changing the locale can change the system font, and thus controls might be resized. If a static control had a bitmap, the bitmap would no longer fit the control. This style bit dictates automatic redimensioning of bitmaps to fit their controls.

  If CENTERIMAGE is specified, the bitmap or icon is centered (and clipped if needed). If CENTERIMAGE is not specified, the bitmap or icon is stretched or shrunk.

  Note that the redimensioning in the two axes are independent, and the result may have a changed aspect ratio.

SIZEIMAGE
: The actual resource width of the image is used.

  SIZEIMAGE is always used in conjunction with ICON. For icon images, the static control is resized accordingly, but the icon remains aligned to the originally specified left and top edges of the control.

  Note that if CENTERIMAGE is also specified, the icon is centered within the control's space rectangle.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

BORDER
: The *border* window style.

GROUP
: The *group* control style.

TAB
: The control is created with the *tabstop* style. By default, static controls do not have the tabstop style.

text [optional]
: The text for the static control. Only static text controls will display the text if it is assigned. If this argument is omitted then the empty string is used.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

Example:

This example adds a static image control and a static text control.

```
::method defineDialog

  self~createStatic(IDC_BMP_PICTURE, 10, 10, 20, 17, "BITMAP REALSIZEIMAGE")
  self~createStatic(IDC_ST_DESCRIPTION, 14, 190, 176, 20, "TEXT LEFT", "Description")
  self~createPushButton(IDC_PB_NEXT, 10, 223, 50, 14, , "Next", onNext)
  self~createPushButton(IDOK, 140, 223, 50, 14, "DEFAULT", "Ok", ok)
```

8.6.13.9. createStaticFrame

```
>>--createStaticFrame(--id--,-x--,-y--,-cx--,-cy--+----------+--)--------------><
                                                  +-,-style--+
```

Creates one of the frame, rectangle, or etched line *static* controls in the dialog template. The *style* option dictates which type is created.

Arguments:

The arguments are:

id [optional]
: The *resource ID* for the static control. -1 is used if *id* is omitted.

x, y, cx, cy [required]

The control *coordinates*

style [optional]

A list of 0 or more *style* keywords separated by spaces. If this argument is omitted, the default is WHITERECT.

The style keywords, WHITERECT, GRAYRECT, BLACKRECT, WHITEFRAME, GRAYFRAME, BLACKFRAME, ETCHED, HORZ, VERT, and LEFT are all mutually exclusive. They determine which frame, rectangle, or line is created. If more than one of these keywords is specified, the type of static frame created is undefined.

WHITERECT	ETCHED	HIDDEN
GRAYRECT	HORZ	DISABLED
BLACKRECT	VERT	GROUP
WHITEFRAME	LEFT	BORDER
GRAYFRAME	NOTIFY	TAB
BLACKFRAME	SUNKEN	

WHITERECT

A *rectangle*)rectangle filled with the current window background color, which is white in the default color scheme.

WHITEFRAME

A *frame* drawn with the same color as the window background, which is white in the default color scheme.

GRAYRECT

A *rectangle*)rectangle filled with the current screen background color, which is gray in the default color scheme.

GRAYFRAME

A *frame* drawn with the same color as the current screen background, which is gray in the default color scheme.

BLACKRECT

A *rectangle*)rectangle filled with the current window frame color, which is black in the default color scheme.

BLACKFRAME

A *frame* drawn with the same color as the current window frames, which is black in the default color scheme.

ETCHED

A *frame* drawn with an etched appearance.

HORZ

A *frame* drawn with an etched appearance, but only the top horizontal line of the frame is drawn. In other words, this is a single horizontal line, whose position and length are determined by the position and width of the imaginary frame specified by the control coordinates.

VERT

A *frame* drawn with an etched appearance, but only the left vertical line of the frame is drawn. In other words, this is a single vertical line, whose position and length are determined by the position and height of the imaginary frame specified by the control coordinates.

NOTIFY

Causes the static control to send notification messages for the click, double click, disabled, and enabled events. A static control without this style does not send notification messages. See the *connectStaticEvent* method of the *EventNotification* class for further information.

SUNKEN

The control is drawn with a half-sunken border around it.

HIDDEN

The not *visible* window style.

DISABLED

The *disabled* window style.

BORDER

The *border* window style.

GROUP

The *group* control style.

TAB

The *tabstop* control style.

Return value:

The possible return values are:

-2

The dialog template has either not been started or is complete.

-1

A symbolic ID was used, but it could not be resolved correctly.

0

Success.

8.6.13.10. createStaticImage

```
>>--createStaticImage(--id--,-x--,-y--,-cx--,-cy--+----------+--)--------------><
                                                  +-,-style--+
```

The *createStaticImage* method creates a static control that displays images in the dialog template.

Arguments:

The arguments are:

id [required]

The *resource ID* for the static control. The resource id is required for this method because, without a positive resource ID, there is no way to set the image in the control.

x, y, cx, cy [required]

The control *coordinates*

style [optional]

A list of 0 or more *style* keywords separated by spaces. If this argument is omitted, the default is ICON.

BITMAP	CENTERIMAGE	DISABLED
METAFILE	RIGHTJUST	GROUP
ICON	SIZECONTROL	BORDER
NOTIFY	SIZEIMAGE	TAB
SUNKEN	HIDDEN	

BITMAP

A static image control is created that will use a bitmap. If neither BITMAP nor METAFILE are specified, the control will be an ICON image control.

METAFILE

A static image control that uses an enhanced metafile will be created. If neither BITMAP nor METAFILE are specified, the control will be an ICON image control.

ICON

A static image control that uses an icon will be created. Cursors are also icons. This is the default.

NOTIFY

Causes the static control to send notification messages for the click, double click, disabled, and enabled events. A static control without this style does not send notification messages. See the *connectStaticEvent* method of the *EventNotification* class for further information.

SUNKEN

The control is drawn with a half-sunken border around it.

CENTERIMAGE

Bitmaps are centered in the static control that contains it. The control is not resized, so that a bitmap too large for the control will be clipped.

RIGHTJUST

The lower right corner of the static control with the BITMAP or ICON style remains fixed when the control is resized. Only the top and left sides are adjusted to accommodate a new bitmap or icon.

SIZECONTROL

Adjusts the bitmap to fit the size of the static control. For example, changing the locale can change the system font, and thus controls might be resized. If a static control had a bitmap, the bitmap would no longer fit the control. This style bit dictates automatic redimensioning of bitmaps to fit their controls.

If CENTERIMAGE is specified, the bitmap or icon is centered (and clipped if needed). If CENTERIMAGE is not specified, the bitmap or icon is stretched or shrunk.

Note that the redimensioning in the two axes are independent, and the result may have a changed aspect ratio.

SIZEIMAGE

The actual resource width of the image is used.

SIZEIMAGE is always used in conjunction with ICON. For icon images, the static control is resized accordingly, but the icon remains aligned to the originally specified left and top edges of the control.

Note that if CENTERIMAGE is also specified, the icon is centered within the control's space rectangle.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

BORDER
: The *border* window style.

GROUP
: The *group* control style.

TAB
: The *tabstop* control style.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

Remarks:

After adding the static image control, the programmer will still need to assign an image to the control. This is done with either the *setImage* method or the *setIcon* methods of the static control. This can only be done once the underlying dialog has been created, meaning in*initDialog* or later.

Example:

This example adds a static image control that uses a bitmap and the control automatically sizes itself to the size of the bitmap.

```
::method defineDialog

  self~createStaticImage(IDC_BMP_PICTURE, 10, 10, 20, 17, "BITMAP REALSIZEIMAGE")
  self~createStatic(IDC_ST_DESCRIPTION, 14, 190, 176, 20, "TEXT LEFT", "Description")
  self~createPushButton(IDC_PB_NEXT, 10, 223, 50, 14, , "Next", onNext)
  self~createPushButton(IDOK, 140, 223, 50, 14, "DEFAULT", "Ok")
```

8.6.13.11. createStaticText

```
>>--createStaticText(-+------+--x-,--y-+------+-+------+-+----------+-+--------+-)-><
                      +-id-,-+         +-,-cx-+ +-,-cy-+ +-,-style--+ +-,-text-+
```

The *createStaticText* method creates a static text control in the dialog template.

Arguments:

The arguments are:

id [optional]

The *resource ID* for the static control.

x [required], y [required], cx [optional], cy [optional]

The control *coordinates*. When cx and cy are omitted, then width and height of the control are calculated automatically based on the width and height of the text string.

style [optional]

A list of 0 or more *style* keywords separated by spaces. If this argument is omitted LEFT is the default.

TEXT	CENTERIMAGE	HIDDEN
LEFT	SUNKEN	DISABLED
CENTER	EDITCONTROL	GROUP
RIGHT	ENDELLIPSIS	BORDER
SIMPLE	NOPREFIX	TAB
LEFTNOWRAP	PATHELLIPSIS	
NOTIFY	WORDELLIPSIS	

LEFT

Left aligns the text. Lines are automatically word wrapped. Words longer than the width of the control are truncated. This is the default if the style argument is omitted.

CENTER

Center aligns the text alignment. Lines are automatically word wrapped. Words longer than the width of the control are truncated. If no text alignment style is specified, the text is left aligned.

RIGHT

Right aligns the text. Lines are automatically word wrapped. Words longer than the width of the control are truncated. If no text alignment style is specified, the text is left aligned.

SIMPLE

A single line of left-aligned text is used. The text line cannot be shortened or altered in any way. If the control is disabled, the text is not grayed.

LEFTNOWRAP

A single line of text, left-aligned is used. Tabs are expanded, but words are not wrapped. Text that extends past the end of a line is clipped.

NOTIFY

Causes the static control to send notification messages for the click, double click, disabled, and enabled events. A static control without this style does not send notification messages. See the *connectStaticEvent* method of the *EventNotification* class for further information.

CENTERIMAGE

The text is centered vertically in the client area of the control.

SUNKEN

The control is drawn with a half-sunken border around it.

EDITCONTROL
: The static control will act like a multi-line edit control for the text it displays. This consists of calculating the average character width in the same manner as the edit control, and not displaying a partially visible last line.

ENDELLIPSIS
: If the text does not fit in the rectangle, it is truncated and an ellipsis is added. Compare this to PATHELLIPSIS.

NOPREFIX
: Any ampersand (&) characters in the control's text are not interpreted as accelerator prefix characters. The text is displayed with the ampersand removed and the next character in the text underlined.

PATHELLIPSIS
: If the text is too big for the specified rectangle, characters in the middle of the text are replaced with an ellipsis so that the result fits in the rectangle. If the text contains backslash (\) characters, this style preserves as much as possible of the text after the last backslash.

WORDELLIPSIS
: Any word that does not fit in the rectangle is truncated and ellipses are added.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

BORDER
: The *border* window style.

GROUP
: The *group* control style.

TAB
: The *tabstop* control style.

text [optional]
: The text for the static control. If this argument is omitted then the empty string is used.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

Example:

This example is similar to the one used to produce the *picture* of frames and rectangles. You can copy and paste it into 2 files and it should run as is.

```
/* UserAllStyles.rex  Simple Dialog to display static frames */

  dlg = .SimpleDialog~new( , 'AllStyles.h')
  if dlg~initCode = 0 then do
    dlg~createCenter(269, 183, "Frames and Rectangles", "", , "MS Shell Dlg 2", 8)
    dlg~execute("SHOWTOP", IDI_DLG_OOREXX)
  end

return 0
-- End of entry point.

::class SimpleDialog subclass UserDialog

::method defineDialog

  self~createStaticText( , 56, 6, 20, 8, , "White")
  self~createStaticText( , 100, 6, 19, 8, , "Gray")
  self~createStaticText( , 143, 6, 19, 8, , "Black")

  self~createStaticText( , 9, 31, 33, 8, , "Rectangle")
  self~createWhiteRect(IDC_ST_WHITERECT, 56, 25, 25, 20, "NOTIFY")
  self~createGrayRect(IDC_ST_GRAYRECT, 100, 25, 25, 20)
  self~createBlackRect(IDC_ST_BLACKRECT, 143, 25, 25, 20)

  self~createStaticText( , 9, 64, 33, 8, , "Frame")
  self~createWhiteFrame(IDC_ST_WHITEFRAME, 56, 58, 25, 20, "NOTIFY")
  self~createGrayFrame(IDC_ST_GRAYFRAME, 100, 58, 25, 20)
  self~createBlackFrame(IDC_ST_BLACKFRAME, 143, 58, 25, 20)

  self~createStaticText( , 9, 94, 46, 8, , "Sunken")
  self~createWhiteFrame(IDC_ST_WHITEFRAME_SUNKEN, 56, 88, 25, 20, "NOTIFY SUNKEN")
  self~createGrayFrame(IDC_ST_GRAYFRAME_SUNKEN, 100, 88, 25, 20, "SUNKEN")
  self~createBlackFrame(IDC_ST_BLACKFRAME_SUNKEN143, 88, 25, 20, "SUNKEN")

  self~createStaticText( , 9, 125, 46, 8, , "Border")
  self~createWhiteFrame(IDC_ST_WHITEFRAME_BORDER, 56, 119, 25, 20, "NOTIFY BORDER")
  self~createGrayFrame(IDC_ST_GRAYFRAME_BORDER, 100, 119, 25, 20, "BORDER")
  self~createBlackFrame(IDC_ST_BLACKFRAME_BORDER143, 119, 25, 20, "BORDER")

  self~createStaticText( , 9, 151, 33, 20, "CENTER", "Sunken / Border")
  self~createWhiteFrame(IDC_ST_WHITEFRAME_SUNKEN_BORDER, 56, 151, 25, 20, "NOTIFY SUNKEN
 BORDER")
  self~createGrayFrame(IDC_ST_GRAYFRAME_SUNKEN_BORDER, 100, 151, 25, 20, "SUNKEN BORDER")
  self~createBlackFrame(IDC_ST_BLACKFRAME_SUNKEN_BORDER, 143, 151, 25, 20, "SUNKEN
 BORDER")

  self~createStaticText( , 193, 31, 23, 8, , "Etched")
  self~createStaticText( , 193, 59, 31, 17, , "Etched Horizontal")
  self~createStaticText( , 193, 89, 31, 17, , "Etched Vertical")
  self~createEtchedFrame(IDC_ST_ETCHED, 234, 25, 25, 20, "NOTIFY")
  self~createEtchedHorizontal(IDC_ST_HORZ, 234, 58, 25, 20, "NOTIFY")
  self~createEtchedVertical(IDC_ST_VERT, 234, 88, 25, 20, "NOTIFY")

  self~createPushButton(IDOK, 210, 157, 50, 14, "DEFAULT", "Ok")

  self~connectStaticEvent(IDC_ST_WHITERECT, "CLICK", onClick)
  self~connectStaticEvent(IDC_ST_WHITEFRAME, "CLICK", onClick)
  self~connectStaticEvent(IDC_ST_WHITEFRAME_SUNKEN, "CLICK", onClick)
  self~connectStaticEvent(IDC_ST_WHITEFRAME_BORDER, "CLICK", onClick)
  self~connectStaticEvent(IDC_ST_WHITEFRAME_SUNKEN_BORDER, "CLICK", onClick)
  self~connectStaticEvent(IDC_ST_ETCHED, "CLICK", onClick)
  self~connectStaticEvent(IDC_ST_HORZ, "CLICK", onClick)
  self~connectStaticEvent(IDC_ST_VERT, "CLICK", onClick)

::method onClick
  say 'Got click on static with ID:' .DlgUtil~loWord(arg(1))
```

```
/* AllStyles.h */

#define IDD_DIALOG1                         100
#define IDC_ST_WHITERECT                    215
#define IDC_ST_GRAYRECT                    1001
#define IDC_ST_BLACKRECT                   1002
#define IDC_ST_ETCHED                      1003
#define IDC_ST_WHITEFRAME                  1004
#define IDC_ST_GRAYFRAME                   1005
#define IDC_ST_BLACKFRAME                  1006
#define IDC_ST_HORZ                        1007
#define IDC_ST_WHITEFRAME_SUNKEN           1008
#define IDC_ST_GRAYFRAME_SUNKEN            1009
#define IDC_ST_BLACKFRAME_SUNKEN           1010
#define IDC_ST_VERT                        1011
#define IDC_ST_WHITEFRAME_BORDER           1012
#define IDC_ST_GRAYFRAME_BORDER            1013
#define IDC_ST_BLACKFRAME_BORDER           1014
#define IDC_ST_WHITEFRAME_SUNKEN_BORDER    1015
#define IDC_ST_GRAYFRAME_SUNKEN_BORDER     1016
#define IDC_ST_BLACKFRAME_SUNKEN_BORDER    1017
```

8.6.13.12. createWhiteFrame

```
>>--createWhiteFrame(--+------+--x-,-y-,-cx-,-cy--+----------+-)-------------><
                       +--id--+                    +-,--style--+
```

The *createWhiteFrame* method creates a white *frame* static control in the dialog template. The frame is drawn with the current window background color, which is white in the default color scheme.

Arguments:

The arguments are:

id [optional]

The *resource ID* for the static control.

x, y, cx, cy [required]

The control *coordinates*

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces.

NOTIFY	DISABLED	TAB
SUNKEN	GROUP	
HIDDEN	BORDER	

NOTIFY

Causes the static control to send notification messages for the click, double click, disabled, and enabled events. A static control without this style does not send notification messages. See the *connectStaticEvent* method of the *EventNotification* class for further information.

SUNKEN

The control is drawn with a half-sunken border around it.

HIDDEN

The not *visible* window style.

DISABLED
: The *disabled* window style.

BORDER
: The *border* window style.

GROUP
: The *group* control style.

TAB
: The *tabstop* control style.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

Example:

See the Frames and Rectangles *example* .

8.6.13.13. createWhiteRect

```
>>--createWhiteRect(--+------+--x-,-y-,-cx-,-cy--+-----------+-)---------------><
                      +--id--+                   +-,--style--+
```

The *createWhiteRect* method creates a white *rectangle*) static control in the dialog template. The rectangle is filled with the current window background color, which is white in the default color scheme.

Arguments:

The arguments are:

id [optional]
: The *resource ID* for the static control.

x, y, cx, cy [required]
: The control *coordinates*

style [optional]
: A list of 0 or more of the following *style* keywords separated by spaces.

NOTIFY	DISABLED	TAB
SUNKEN	GROUP	
HIDDEN	BORDER	

NOTIFY
: Causes the static control to send notification messages for the click, double click, disabled, and enabled events. A static control without this style does not send notification messages. See the *connectStaticEvent* method of the *EventNotification* class for further information.

SUNKEN
: The control is drawn with a half-sunken border around it.

HIDDEN
: The not *visible* window style.

DISABLED
: The *disabled* window style.

BORDER
: The *border* window style.

GROUP
: The *group* control style.

TAB
: The *tabstop* control style.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

Example:

See the Frames and Rectangles *example* .

8.6.14. createStatusBar

```
>>--createStatusBar(-id-,--x-,--y-,--cx-,--cy-+---------+--+-----------------+--)-><
                                              +-,-style-+  +-,-attributeName-+
```

Creates a *StatusBar* control in the dialog template and specifies the resource ID, position, size, and style name for the control.

Arguments:

The arguments are:

id [required]
: The *resource ID* for the status bar control. May be numeric or *symbolic*.

x, y, cx, cy [required]
: The control *coordinates*. Status bar controls typically ignore the coordinate arguments. Rather they size and position themselves according to their internal logic.

style [optional]
: A list of 0 or more of the following *style* keywords separated by spaces. Case is not significant:

SIZEGRIP	DISABLED	TAB
TOOLTIPS	GROUP	

BORDER HIDDEN

SIZEGRIP
: The status bar control includes a sizing grip at the right end of the status bar. A sizing grip is like a sizing border, it is a rectangular area that the user can click and drag to resize the status bar.

TOOLTIPS
: This style enables ToolTips.

BORDER
: Status bars usually do not have The *border* window style, they draw their own border. The BORDER adds an extra border to the status bar.

DISABLED
: Adds the *disabled* window style.

GROUP
: The status bar will have the *group* control style. By default the status bar does not have the group style.

HIDDEN
: The not *visible* window style.

TAB
: Create the control with the *tabstop* control style. By default the control is created without the tabstop style.

In addition, *createStatusBar* has support for any *Common Control Styles* keyword.

attributeName [optional]
: Although this argument is accepted so that the format of the *createStatusBar* method is similar to the other *createXX* methods, the concept of a status bar having *data* makes no sense. If the argument is used it is ignored.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

Remarks:

Experimentation has shown that the CCS_LEFT, CCS_TOP, and CCS_RIGHT styles can be used with the status bar control. Although, left and right styles do not display the text of the status bar very well.

Details

Raises syntax errors when incorrect usage is detected.

Example:

The following example creates a status bar in the dialog template with both the sizing grip and tool tip styles. Note that the position and size of the status bar are ignored by the operating system when it creates the control. Instead, the control is placed at the bottom of the dialog, using a default height, and its width is the width of the dialog:

```
self~createStatusBar(IDC_STATUS, 167, 90, 50, 14, 'SIZEGRIP TOOLTIPS')
```

8.6.15. createTab

```
>>--createTab(-id-,--x-,--y-,--cx-,--cy-+---------+--+-----------------+--)----><
                                        +-,-style-+  +-,-attributeName-+
```

Creates a *Tab* control in the dialog template and specifies the resource ID, position, size, style, and data attribute name for the control.

Arguments:

The arguments are:

id [required]

The *resource ID* for the tab control. May be numeric or *symbolic*.

x, y, cx, cy [required]

The control *coordinates*

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces:

ALIGNRIGHT	FOCUSONDOWN	DISABLED
BUTTONS	ICONLEFT	GROUP
CLIPSIBLINGS	LABELLEFT	BORDER
FIXED	MULTILINE	TAB
FOCUSNEVER	HIDDEN	

ALIGNRIGHT

The width of each tab is increased, only when needed, so that each row of tabs fills the entire width of the tab control. This style is ignored unless the MULTILINE style is also specified.

BUTTONS

Makes the tabs look like buttons. The tabs should serve the same function as button controls. Clicking a tab should carry out a command instead of displaying a page. Because the display area in a button tab control is typically not used, no border is drawn around it.

CLIPSIBLINGS

This is a general window style rather than a specific tab control style. Child windows are clipped relative to each other. This effects how the operating system draws the window. Microsoft explains it in this way: when a child window receives a paint message, the CLIPSIBLINGS style clips all other overlapped child windows out of the region of the child window to be updated. When a window does not have the CLIPSIBLINGS and child windows overlap, when drawing is done within the client area of a child window, it is possible that the drawing will take place within the client area of one of the other overlapped windows.

FIXED
: All tabs are the same width. This style cannot be combined with the ALIGNRIGHT style.

FOCUSNEVER
: When clicking a tab, it does not receive the input focus.

FOCUSONDOWN
: When clicking a tab, it does receive the input focus.

ICONLEFT
: Icons are aligned with the left edge of each fixed-width tab. This style can only be used with the FIXED style.

LABELLEFT
: The label is displayed immediately to the right of the icon instead of being centered. This style can only be used with the FIXED style, and it implies the ICONLEFT style.

MULTILINE
: Tabs are displayed in multiple rows when there are more tabs than can be displayed in a single row. In this way all tabs are visible.

HIDDEN
: The control has the not *visible* window style.

DISABLED
: The control has the *disabled* window style.

GROUP
: The control has *group* control style.

BORDER
: Create the control with the *border* window style. By default tab controls are created without the border style.

NOTTAB
: Create the control without the *tabstop* control style. By default the control is created with the tabstop style.

attributeName [optional]
: The name of the data *attribute* associated with the dialog control. If you omit this argument then the name of the data attribute is constructed *automatically*. If *automatic* data detection is off, this argument is ignored.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

1
: The control was added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

Example:

The following example creates a tab control within the in-memory dialog template. Its resource id is the numeric value of the *symbolic* id: IDC_TAB. If needed, multiple rows of tabs are displayed so tat all tabs are visible at once. All tabs have the same width. Its data (the selected tab) is associated with data attribute CURRENTPAGE.

```
self~createTab(IDC_tab, 10, 120, 200, 100, "MULTILINE FIXED", "CURRENTPAGE")
```

8.6.16. createToolBar

```
>>--createToolBar(-id-,--x-,--y-,--cx-,--cy-+---------+--+-----------------+--)-><
                                             +-,-style-+  +-,-attributeName-+
```

Creates a *ToolBar* control in the dialog template and specifies the resource ID, position, size, and style for the control.

Arguments:

The arguments are:

id [required]

The *resource ID* for the toolbar control. May be numeric or *symbolic*.

x, y, cx, cy [required]

The control *coordinates*. Toolbar controls typically ignore the coordinate arguments. Rather they size and position themselves according to their internal logic.

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces:

ALTDRAG	TOOLTIPS	GROUP
CUSTOMERASE	TRANSPARENT	HIDDEN
FLAT	WRAPABLE	NOTAB
LIST	NOBORDER	
REGISTERDROP	DISABLED	

ALTDRAG

Allows users to change a toolbar button's position by dragging it while holding down the ALT key. If this style is not specified, the user must hold down the SHIFT key while dragging a button. Note that the CCS_ADJUSTABLE style must be specified to enable toolbar buttons to be dragged.

CUSTOMERASE

Generates CUSTOMDRAW notification messages when the toolbar processes WM_ERASEBKGND messages. Note that support for toolbar *CustomDraw* has not yet been implemented in the ooDialog framework.

FLAT

Creates a flat toolbar. In a flat toolbar, both the toolbar and the buttons are transparent and hot-tracking is enabled. Button text appears under button bitmaps.

LIST
: Creates a flat toolbar with button text to the right of the bitmap. Otherwise, this style is identical to FLAT.

REGISTERDROP
: Generates GETOBJECT notification messages to request drop target objects when the cursor passes over toolbar buttons. Although the ooDialog framework supports this style and event notification, there is no way within the framework to make use of it.

TOOLTIPS
: Creates a ToolTip control that an application can use to display descriptive text for the buttons in the toolbar.

TRANSPARENT
: Creates a transparent toolbar. In a transparent toolbar, the toolbar is transparent but the buttons are not. Button text appears under button bitmaps.

WRAPABLE
: Creates a toolbar that can have multiple lines of buttons. Toolbar buttons can "wrap" to the next line when the toolbar becomes too narrow to include all buttons on the same line. When the toolbar is wrapped, the break will occur on either the rightmost separator or the rightmost button if there are no separators on the bar. This style must be set to display a vertical toolbar control when the toolbar is part of a vertical rebar control. This style cannot be combined with CCS_VERT.

NOBORDER
: By default a toolbar has the *border* window style. Use the NOBORDER keyword for a toolbar without a border

DISABLED
: The toolbar will have the *disabled* window style. By default the toolbar is enabled.

GROUP
: The toolbar will have the *group* control style. By default toolbars are not given the group style.

HIDDEN
: By default the toolbar is created with the *visible* window style. Use the HIDDEN keyword to have the toolbar created invisible.

NOTAB
: Create the control without the *tabstop* control style. By default the control is created with the tabstop style.

In addition, the *createToolBar* method has support for any *Common Control Styles* keyword.

attributeName [optional]
: Although this argument is accepted so that the format of the *createToolBar* method is similar to the other *createXX* methods, the concept of a toolbar having *data* makes no sense. If the argument is used it is ignored.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
    A symbolic ID was used, but it could not be resolved correctly.

0
    Success.

Remarks:
Additional comments.

Details
Raises syntax errors when incorrect usage is detected.

Example:
The following example creates a toolbar that is initially invisible. In the application, after the buttons from the toolbar have been added, the toolbar is made visible:

```
 self~createToolBar(IDC_TOOLBAR, 0, 0, 20, 11, "TOOLTIPS LIST HIDDEN BORDER FLAT
CCS_ADJUSTABLE ALTDRAG")
```

8.6.17. createTrackbar

```
>>--createTrackBar(-id-,--x-,--y-,--cx-,--cy-+---------+--+------------+--)---->< 
                                             +-,-style-+  +-,-attrName-+
```

Creates a *TrackBar* control in the dialog template and specifies the resource ID, position, size, style, and attribute name for the control.

Arguments:
The arguments are:

id [required]
    The *resource ID* for the trackbar control. May be numeric or *symbolic*.

x, y, cx, cy [required]
    The control *coordinates*

style [optional]
    A list of 0 or more of the following *style* keywords separated by spaces:

AUTOTICKS	BOTTOM	HIDDEN
NOTICKS	LEFT	DISABLED
HORIZONTAL	RIGHT	GROUP
VERTICAL	BOTH	NOBORDER
TOP	ENABLESELRANGE	NOTAB

AUTOTICKS
    Creates a trackbar that has a tick mark for each increment in its range of values. These tick marks are automatically added when an application calls the *initRange* method. You cannot use the *setTickAt* or *setTickFrequency* methods to specify the position of the tick marks when you use this option.

NOTICKS
    Creates a trackbar that does not display tick marks.

HORIZONTAL
: Orients the trackbar horizontally. This is the default orientation.

VERTICAL
: Orients the trackbar vertically.

TOP
: Displays tick marks along the top of a horizontal trackbar.

BOTTOM
: Displays tick marks along the bottom of a horizontal trackbar. This option can be used together with the TOP option to display tick marks on both sides of the trackbar control.

LEFT
: Displays tick marks along the left of a vertical trackbar.

RIGHT
: Displays tick marks along the right of a vertical trackbar. This option can be used together with the LEFT option to display tick marks on both sides of the trackbar control.

BOTH
: Displays tick marks on both sides of the trackbar in any direction.

ENABLESELRANGE
: Displays a selection range. If you set this option, the tick marks at the starting and ending positions of a selection range are displayed as triangles and the selection range is highlighted. This can be used, for example, to indicate a preferred selection.

HIDDEN
: The trackbar is created with the not *visible* window style.

DISABLED
: The trackbar is created with the *disabled* window style.

GROUP
: The trackbar is created with the *group* control style.

BORDER
: Crate the trackbar with the *border* window style. Normally trackbars are not created with the border style.

NOTAB
: Create the trackbar without the *tabstop* control style. By default the trackbar is created with the tabstop style.

attrName [optional]
: The name of the data *attribute* associated with the dialog control. If you omit this argument then the name of the data attribute is constructed *automatically*. If *automatic* data detection is off, this argument is ignored.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

1
: The control was added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

Example:

The following example creates a vertical trackbar at the right border of the dialog. The trackbar has the *symbolic* resource ID of IDC_TB_PRESSURE and its data *attribute* is named PRESSURE. The trackbar displays automatic tick marks on both sides.

```
self~createTrackbar(IDC_TB_PRESSURE, self~sizeX - 30, 15, 20, self~sizeY - 30, -
                    "VERTICAL BOTH AUTOTICKS", "PRESSURE")
```

8.6.18. createTreeView

```
>>--createTreeView(-id-,--x-,--y-,--cx-,--cy-+---------+--+------------+--)----><
                                             +-,-style-+  +-,-attrName-+
```

Creates a tree *TreeView* control in the dialog template and specifies the resource ID, position, size, style, and attribute name for the control.

Arguments:

The arguments are:

id [required]
: The *resource ID* for the tree view control. May be numeric or *symbolic*.

x, y, cx, cy [required]
: The tree view *coordinates*.

style [optional]
: A list of 0 or more of the following *style* keywords separated by spaces:

ATROOT	NODRAG	TRACKSELECT
BUTTONS	NOHSCROLL	VSCROLL
CHECKBOXES	NONEVENHEIGHT	ALL
EDIT	NOSCROLL	HIDDEN
FULLROWSELECT	NOTOOLTIPS	DISABLED
HSCROLL	RTLREADING	GROUP
INFOTIP	SHOWSELALWAYS	NOBORDER
LINES	SINGLEEXPAND	NOTAB

ATROOT
: The tree-view has lines linking child items to the root of the hierarchy.

BUTTONS
: Displays plus (+) and minus (-) buttons next to parent items. The user clicks the buttons to expand or collapse a parent item's list of child items. To include buttons with items at the root of the tree view, the LINES style must also be specified.

CHECKBOXES

Displays a check box with the item. In earlier versions of Windows, the check box would only be displayed if the tree-view item had an image associated with it. If ooDialog is running on Windows XP or later this earlier behaviour no longer applies.

EDIT

With this style, the tree-view allows the user to edit the tree-view item labels. To have an edited label take effect, the programmer must connect an *event* notification using the *connectTreeViewEvent* method. For the most flexibility with edit labeling, connect both the *BEGINEDIT* and *ENDEDIT* notifications. Alternatively the *DEFAULTEDIT* event could be connected using the *connectListViewEvent* method.

FULLROWSELECT

Enables full-row selection in the tree view. The entire row of the selected item is highlighted, and clicking anywhere on an item's row causes it to be selected. This style cannot be used in conjunction with the LINES style.

HSCROLL

The tree view supports a horizontal scroll bar.

INFOTIP

The tree-view will request info tip text by sending the *GETINFOTIP* notification. If the tree-view does not have the INFOTIP style, GETINFOTIP notifications are not sent.

LINES

The tree-view has lines linking child items to their corresponding parent items.

NODRAG

The tree-view is prevented from sending *BEGINDRAG* notifications.

NOHSCROLL

Disables horizontal scrolling in the control. The control will not display any horizontal scroll bars.

NONEVENHEIGHT

Allows setting the height of the items to an odd height with the *setItemHeight*. By default, the height of items must be an even value.

NOSCROLL

Disables both horizontal and vertical scrolling in the tree-view. The control will not display any scroll bars.

NOTOOLTIPS

By default, a tree-view always creates a child ToolTip. This style prevents the creation of a ToolTip.

RTLREADING

Causes text to be displayed from right-to-left (RTL). Usually, windows display text left-to-right (LTR). Windows can be mirrored to display languages such as Hebrew or Arabic that read RTL. Typically, tree-view text is displayed in the same direction as the text in its parent window. If RTLREADING is set, tree-view text reads in the opposite direction from the text in the parent window.

SHOWSELALWAYS

A selected item remains selected when the tree view loses focus.

SINGLEEXPAND
: Causes the item being selected to expand and the item being unselected to collapse upon selection in the tree view. If the user holds down the CTRL key while selecting an item, the item being unselected will not be collapsed.

TRACKSELECT
: Enables hot tracking in a tree-view control.

VSCROLL
: The tree view supports a vertical scroll bar.

ALL
: The styles ATROOT, BUTTONS, EDIT, HSCROLL, LINES, SHOWSELALWAYS, and VSCROLL are all applied.

HIDDEN
: The tree view has the not *visible* window style.

DISABLED
: The tree view has the *disabled* window style.

GROUP
: The tree view has the *group* control style.

NOBORDER
: The tree view is created without the *border* style. Normally the tree view is created with the border style.

NOTAB
: Create the control without the *tabstop* control style. By default the control is created with the tabstop style.

attributeName [optional]
: The name of the data *attribute* associated with the tree view control. If you omit this argument then the name of the data attribute is constructed *automatically*. If *automatic* data detection is off, this argument is ignored.

Return value:

The possible return values are:

-2
: The dialog template has either not been started or is complete.

-1
: A symbolic ID was used, but it could not be resolved correctly.

0
: Success.

1
: The control was added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

Example:

The following example creates a tree view with the *symbolic* resource ID. The control is positioned at x=100 and y=80 and with a size of width=40 and height=120. The name for the data attribute of the tree view will be ORGCHART The tree view uses lines for child items and roots, displays a button to the left of each parent item, and supports item editing.

```
self~createTreeView(IDC_TV_ORG_CHART, 100, 80, 40, 120, "LINES BUTTON EDIT ATROOT",
 "ORGCHART")
```

8.6.19. createUpDown

```
>>--createUpDown(-id-,--x-,--y-,--cx-,--cy-+---------+--+-----------------+--)-><
                                           +-,-style-+  +-,-attributeName-+
```

Creates an *UpDown* control in the dialog template and specifies the resource ID, position, size, style, and attribute name for the control.

Arguments:

The arguments are:

id [required]

The *resource ID* for the up-down control. May be numeric or *symbolic*.

x, y, cx, cy [required]

The control *coordinates*

style [optional]

A list of 0 or more of the following *style* keywords separated by spaces:

LEFT	HOTTRACK	DISABLED
RiGHT	NOTHOUSANDS	GROUP
ARROWKEYS	BUDDYINT	BORDER
AUTOBUDDY	WRAP	TAB
HORIZONTAL	HIDDEN	

LEFT

Positions the up-down control next to the left edge of the buddy window. The buddy window is moved to the right, and its width is decreased to accommodate the width of the up-down control. The default style is the RIGHT style if neither LEFT nor RIGHT are specified.

RIGHT

Positions the up-down control next to the right edge of the buddy window. The width of the buddy window is decreased to accommodate the width of the up-down control. This is the default style if neither LEFT nor RIGHT are specified.

ARROWKEYS

With this style the up-down control will increment or decrement the position of the control when the UP ARROW and DOWN ARROW keys are pressed and the control has the focus.

AUTOBUDDY

With this style, the up-down control automatically selects the previous window in the z-order as the up-down control's buddy window. In the ooDialog framework, the previous window would normally be the dialog control window created immediately before the up-down control. Although, the z-order can be changed using the *setWindowPos* or *moveWindow* methods.

HORIZONTAL

The up-down control's arrows normally point up and down. The style changes that so the arrows point left and right instead.

HOTTRACK

Hot tracking behavior has the up-down control highlight the UP and DOWN arrows on the control as the mouse pointer passes over them.

NOTHOUSANDS

The thousands separator is not inserted between every three decimal digits.

BUDDYINT

The up-down control will set the text of the buddy window when the position in the control changes. The text consists of the position formatted as a decimal or hexadecimal string.

WRAP

The position will wrap around to the ending or beginning if it is incremented or decremented beyond the range of the up-down control.

HIDDEN

The not *visible* window style.

DISABLED

The *disabled* window style.

GROUP

The *group* control style.

BORDER

The *border* window style.

TAB

Create the control with the *tabstop* control style. By default the control is created without the tabstop style.

attributeName [optional]

The name of the data *attribute* associated with the dialog control. If you omit this argument then the name of the data attribute is constructed *automatically*. If *automatic* data detection is off, this argument is ignored.

Return value:

The possible return values are:

-2

The dialog template has either not been started or is complete.

-1

A symbolic ID was used, but it could not be resolved correctly.

0

Success.

1

The control was added to the dialog template but the data attribute could not be connected because the maximum number of data connections has been reached.

Example:

The following example creates an up down control with the *symbolic* ID of IDC_UPD positioned at (81, 26) of the dialog's *client area*. The control will be 12 *dialog unit* wide and 16 dialog units high. The up down will wrap around when it reaches either end of its range, the up and down arrow keys can be used to change the position, and the control automatically selects the control previous to to be its *buddy* window. The control will also set the text of its buddy window to the value of the up down control's position.

```
self~createUpDown(IDC_UPD, 81, 26, 12, 16, "WRAP ARROWKEYS AUTOBUDDY SETBUDDYINT")
```

8.7. defineDialog

```
>>--defineDialog---------------------------------><
```

The *defineDialog* method is invoked automatically by the ooDialog framework. It is designed to be over-ridden in a subclass of **`UserDialog`**. The programmer uses this method to dynamically create the dialog controls for the dialog by using the appropriate *create...* methods. For all practical purposes, all dialog controls of a **`UserDialog`** should be created in this method.

Arguments:

No arguments are passed to this method when the ooDialog framework invokes the method.

Return value:

No return valued is expected. The ooDialog framework ignores any return.

Remarks:

The programmer should **not** invoke this method, the ooDialog framework invokes it at the proper time. Rather, the programmer must over-ride the method in his **`UserDialog`** subclass in order to add dialog controls to the dialog.

It is important to note that the *defineDialog* method is invoked **before** the *underlying* Windows dialog is create. Therefore, all methods requiring the underlying dialog to exist can not be used in the *defineDialog* method.

Example:

This example demonstrates how to create a push *button* and an *edit* control in a dialog:

```
::class MyDialog subclass UserDialog
     .
     .
     .
::method defineDialog
self~createPushButton(401, 20, 235, 40, 15, "NOTAB", "&More...")
self~createEdit(402, 20, 170, 150, 10, "AUTOSCROLLH", EDIT_1)
     .
     .
     .
```

8.8. load

```
>>--load(--rcFile--+-------+--+----------+--+------------+--)----------------><
```

```
                    +-,-id--+  +--,-opts--+  +-,--expected-+
```

The *load* method reads the dialog *template* form a resource *script* file. The method invokes the *loadFrame* and *loadItems* methods to parse the resource script statements create the in-memory form of the dialog template needed by the operating system.

An alternative to using the *load* method directly is to subclass the *RcDialog* instead of the **`UserDialog`**. The **`RcDialog`** handles the details of how and when to invoke the *load* method automatically.

Arguments:

The arguments are:

rcFile [required]

The resource script file name.

id [optional]

The resource ID of the dialog. May be numeric or *symbolic*. Resource scripts often contain more than one dialog definition, the resource ID is used to locate the correct dialog. If the resource script only contains one dialog resource, then this argument can be omitted.

opts [optional]

Zero or more of the following keywords, separated by blanks. Case is not significant:

CENTER

The dialog is positioned in the center of the screen. This keyword would over-ride the position specified in the resource file.

CONNECTBUTTONS

For each push button in the dialog template, the *connectButtonEvent* CLICKED event notification is connected to a method in the Rexx dialog object. The label of the push button is used to automatically generate the method name. All space, tab, ampersand, colon, and plus characters are removed from the label. In addition, if there is a trailing "..." it is also removed. The resulting string is used as the method name.

CONNECTRADIOS

For each radio button in the dialog template, the *connectButtonEvent* CLICKED event notification is connected to a method in the Rexx dialog object. The method name is automatically generated from the label of the radion button. All space, tab, ampersand, colon, and plus characters are removed from the label. In addition, if there is a trailing "..." it is also removed. This string has the characters **`ID`** prepended, and the result is used for the method name.

CONNECTCHECKS

This option connects the check box controls as CONNECTRADIOS connects radio buttons. The method name is generated in the same way as it is for radio buttons.

expected [optional]

This is a value used by the ooDialog framework to estimate the amount of memory to reserve for the dialog template. It serves the same purpose as the *expected* argument the *create* method. The default value is 200. A minimum value should be at least the number of dialog controls in the template.

Return value:

This method returns 0 for success and non-zero on error.

Example:

The following example creates a dialog from the dialog template with a symbolic resource id of **`IDD_DIALOG1`** in the **`dialog1.rc`** resource script file. As the resource script file is being parsed, the CLICKED *event* for all push and radio buttons are connected to a methods in the Rexx dialog object. See the CONNECTBUTTONS and CONNECTRADIOS keyword above for a little more detail on how the method names are constructed.

```

  MyDlg = .UserDialog~new( , 'Dialog1.h')
  MyDlg~load("Dialog1.rc", IDD_DIALOG1, "CONNECTBUTTONS CONNECTRADIOS")
```

8.9. loadFrame

```
>>--loadFrame(--rcFile--+-------+--+----------+--+-------------+--)------------><
                        +-,-id--+  +--,-opts--+  +-,--expected-+
```

The *loadFrame* method starts the creation of the in-memory Windows dialog *template* using a dialog definition from a resource *script* file. It creates the non*client area* of the dialog.

Details

This is a private method. It is invoked automatically by the *load* method. There is seldom a need for the Rexx programmer to invoke this method directly.

Arguments:

The arguments are:

rcFile [required]

The resource script file name.

id [optional]

The resource ID of the dialog. May be numeric or *symbolic*. Resource scripts often contain more than one dialog definition, the resource ID is used to locate the correct dialog. If the resource script only contains one dialog resource, then this argument can be omitted.

opts [optional]

If keyword, CENTER, is contained in the opts string then the dialog is positioned in the center of the screen. This option would over-ride the position specified in the resource file.

expected [optional]

This is a value used by the ooDialog framework to estimate the amount of memory to reserve for the dialog template. It serves the same purpose as the *expected* argument the *create* method. The default value is 200. A minimum value should be at least the number of dialog controls in the template.

Return value:

This method returns 0 for success and non-zero on error.

Example:

The following example over-rides the *load* method, so that it only loads the dialog frame, but none of the dialog controls.

```
::class WindowOnlyDialog subclass UserDialog
     .
     .
     .
```

```
::method load
  return self~loadFrame("Dialog2.rc", 100, "CENTER", 20)
```

8.10. loadItems

```
>>--loadItems(--rcFile--+-------+--+----------+--)----------------------------><
                        +-,-id--+  +--,-opts--+
```

The *loadItems* method finishes the creation of the in-memory Windows dialog *template* using a dialog definition from a resource *script* file. It creates the dialog controls for the dialog. In essence it defines the *client area* of the dialog.

Details

This is a private method. It is normally invoked automatically by the *load* method. There is seldom a need for the Rexx programmer to invoke this method directly.

Arguments:

The arguments are:

rcFile [required]

The resource script file name.

id [optional]

The resource ID of the dialog. May be numeric or *symbolic*. Resource scripts often contain more than one dialog definition, the resource ID is used to locate the correct dialog. If the resource script only contains one dialog resource, then this argument can be omitted.

opts [optional]

Zero or more of the following keywords, separated by blanks. Case is not significant:

CONNECTBUTTONS

For each push button in the dialog template, the *connectButtonEvent* CLICKED event notification is connected to a method in the Rexx dialog object. The label of the push button is used to automatically generate the method name. All space, tab, ampersand, colon, and plus characters are removed from the label. In addition, if there is a trailing "..." it is also removed. The resulting string is used as the method name.

CONNECTRADIOS

For each radio button in the dialog template, the *connectButtonEvent* CLICKED event notification is connected to a method in the Rexx dialog object. The method name is automatically generated from the label of the radion button. All space, tab, ampersand, colon, and plus characters are removed from the label. In addition, if there is a trailing "..." it is also removed. This string has the characters **`ID`** prepended, and the result is used for the method name.

CONNECTCHECKS

This option connects the check box controls as CONNECTRADIOS connects radio buttons. The method name is generated in the same way as it is for radio buttons.

Return value:

This method returns 0 for success and non-zero on error.

Example:

In the following example the dialog is created either with the controls of dialog 200 or dialog 300, depending on the *view* argument:

```
::class MyDialog subclass UserDialog
    ...

::method load
   use strict arg view
   self~loadFrame("Dialog2.rc", 200, "CENTER", 200)
   if view == "special" then
      self~loadItems("Dialog2.rc", 200, "CONNECTBUTTONS")
   else
      self~loadItems("Dialog2.rc", 300, "CONNECTBUTTONS")
```

The Dialog Control Object

This chapter discusses the *dialog control object* in a fashion similar to the *dialog* object. It lists the methods that are common to all dialog controls. In the graphical user interface (GUI) for the Windows operating system both dialogs and dialog controls are windows. Therefore, many of the methods of a dialog control object are the same as the methods of the dialog object. These are the methods that are common to all windows, whether they are dialog windows or dialog control windows.

In the Windows GUI all windows are created with a set of *window styles*. Dialog control windows are created using styles that are common to all windows, (for example the visible style,) and styles specific to the dialog control itself, (for example the multi-line style of the *Edit* control class.) In general the styles of a control fall into three categories: Styles that can only be set when the control is created and then can not be changed afterwards. Styles that can be changed after control creation by sending messages to the control. And, styles that can be changed after the control is created by accessing the control window directly.

The ooDialog programmer chooses the window styles for dialog controls when he defines the dialog, either by using a resource editor for dialogs defined with resource scripts or binary compiled resources, or by using the one of the *create...* methods for a user dialog. After the dialog control has been created, the individual dialog control classes provide methods to change those styles that can be changed, either by sending the proper message to the control or by accessing the control window directly.

9.1. Method Table

The following table lists the class methods, attributes, and instance methods that all dialog controls have in common:

Table 9.1. Dialog Control Object Method Reference

Dialog Control Method	Description
Attributes	
factorX	The horizontal size of one dialog unit in pixels. (Inaccurate.)
factorY	The vertical size of one dialog unit in pixels. (Inaccurate.)
hDlg	Reflects the window handle of the underlying Windows dialog that the dialog control belongs to.
hwnd	The window handle of the dialog control.
id	Reflects the numeric resource ID for the dialog control.
initCode	Meaningless for a dialog control, it is always 0.
oDlg	Reflects the Rexx dialog object that the dialog control belongs to.
pixelCX	The width of the dialog control in pixels.
pixelCY	The height of the dialog control in pixels.
sizeX	The width of the dialog control in dialog units. (Inaccurate.)
sizeY	The height of the dialog control in dialog units. (Inaccurate.)
Instance Methods	
assignFocus	Sets the input focus to this dialog control.
childWindowFromPoint	Determines which, if any, of the child windows belonging to this dialog control contains the specified point.

Dialog Control Method	Description
clear	Clears the client area of the control by painting it with the background brush.
clearRect	Clears the specified rectangular within the client area of this control.
client2screen	Converts a point or points in client-area coordinates of the control to its screen coordinates.
clientRect	Returns a **`Rect`** object containng the dimensions of the control's client area in pixels.
clientToScreen	Converts client-area coordinates of the control to its screen coordinates.
connectCharEvent	Connects a character event notification sent to the control to a method in the Rexx dialog.
connectFKeyPress	Connects all F Key key press events to a method in the Rexx dialog.
connectKeyPress	Connects a key press event notification with a method in the Rexx dialog.
createBrush	Creates a logical brush that has the specified style, color, and pattern.
createFont	Returns a handle to a logical font, the implementation is **incorrect**.
createFontEx	Retrieves a handle to a logical font from the system font manager
createPen	Creates a logical pen that has the specified style, width, and color.
data	Retrieves the *data* (the value) of the underlying control's state.
data =	Sets the state of the underlying dialog control to the *data* (the value) specified.
deleteFont	Deletes a font returned from *createFontEx* or *createFont*.
deleteObject	Deletes a graphic object,
disable	Disables the control
disconnectKeyPress	Disconnects a method in the Rexx dialog from a previously connected key press event.
display	Shows or hides the control.
draw	Redraws the entire client area of the control immediately.
drawAngleArc	Draws a partial circle (arc) and a line connecting the start drawing point with the start of the arc.
drawArc	Draws a circle or ellipse withi the given device context using the active pen.
drawLine	Draws a line within the device context using the active pen.
drawPie	Draws and fills a pie of a circle or ellipse within the given device context using the active brush and pen.
drawPixel	Draws a pixel within the device context.
enable	Enables the control
fillDrawing	Fills in an outline figure within the device context using the active brush.
fillRect	Fills a rectangle using the specified brush within the specified device context.
fontColor	Sets the font color for a device context.
fontToDC	Loads a font into a device context and returns the handle of the previous font.

Dialog Control Method	Description
foregroundWindow	Returns the handle of the window in the foreground.
freeDC	Releases a device context.
getArcDirection	Returns the current drawing direction.
getClientRect	Returns the dimensions of the control's client area.
getDC	Returns the handle to the display device context of a dialog control.
getExStyleRaw	Retrieves the numeric value of the control's extended style flags.
getFont	Returns the font used by the dialog control.
getID	Retrieves the identification number of the control.
getPixel	Returns the color number of a pixel within the device context.
getPos	Returns the position of the control in dialog units **(not accurate.)**
getRealPos	Returns the position of the control in pixels as a **`Point`** object.
getRealSize	Returns the size of the control in pixels as a **`Size`** object.
getRect	Returns the dimensions of the control.
getSize	Returns the size of the control in dialog units **(not accurate.)**
getStyleRaw	Retrieves the numeric value of the control's style flags.
getSysBrush	Retrieves a handle to a logical brush that corresponds to the specified system color index.
getText	Gets the text of the control.
getTextAlign	Gets the text alignment setting for the specified device context.
getTextExtent	Gets the bounding rectangle, as a *Size* object for the specified text, if it were to be drawn in the specified device context.
getTextSizeDlg	Calculates the size, (width and height,) in dialog units for a given string.
getTextSizePx	Calculates the size needed for a string in pixels **(preferred method.)**
getTextSizeScreen	Calculates the size needed for a string in pixels.
group	Adds or removes the *group* style for the control.
hasKeyPressConnection	Queries if a specific method, or any method, in the Rexx dialog has a connection to a key press event.
hide	Makes the control invisible and repaints it.
hideFast	Marks the control as invisible
hScrollPos	Returns the position of the horizontal scroll bar in the dialog control.
isEnabled	Tests if the control is enabled.
isVisible	Tests if the control is visible.
loadBitmap	Loads a bitmap from a file into memory and returns the handle to the bitmap.
mapWindowPoints	Converts, or maps, a set of points from the coordinate space relative to this dialog control to a coordinate space relative to the specified window.
move	Moves the control to the position specified in dialog units **(not accurate.)**
moveTo	Moves the control to the position specified in pixels.
moveWindow	Changes the position, visibility, and Z order of the dialog.

Dialog Control Method	Description
objectToDC	Loads a graphic object into a device context.
opaqueText	Sets the text drawing mode in a device context to opaque, (background is redrawn with the current brush.)
rectangle	Draws a rectangle within the given device context.
redraw	Redraws the entire control window and all its child windows immediately.
redrawClient	Redraws the entire client area of the control immediately.
redrawRect	Immediately redraws the rectangle of the client area of the associated dialog.
removeBitmap	Frees an in-memory bitmap that was loaded through the *loadBitmap* method.
resize	Resizes the control to the size specified in dialog units **(not accurate.)**
resizeTo	Resizes the control to the size specified in pixels.
screen2client	Converts a point or points in screen coordinates to the client-area coordinates of the control.
screenToClient	Converts screen coordinates to the client-area coordinates of the control.
scroll	Scrolls the contents of the dialog control's client area by the amount specified.
sendMessage	Sends a Windows message to the underlying control and returns its response as a whole number.
sendMessageHandle	Sends a Windows message to the underlying control and returns its response as a handle.
setArcDirection	Sets the current drawing direction.
setColor	Sets the background color, and optionally the text color, for this dialog control.
setFont	Sets a new font to be used by the dialog control.
setHScrollPos	Sets the thumb position of the horizontal scroll bar contained in the dialog control.
setParent	Sets a new parent for this dialog control.
setRect	Moves and / or resizes the control.
setStyleRaw	Sets the value of the window's style flags using the numeric value specified.
setSysColor	Sets the background color or foreground color, or both, for this dialog control using the system *colors*
setText	Sets the text of the control.
setTextAlign	Sets the text alignment option for the specified device context.
setTitle	Sets the text of the control.
setVScrollPos	Sets the thumb position of the vertical scroll bar contained in the dialog control.
setWindowPos	Changes the size, position, visibility, and Z order of the dialog.

Dialog Control Method	Description
setWindowTheme	Changes the visual theme for this dialog control window, causes the window to use a different set of visual style information than its class normally uses.
show	Makes the control visible if it was hidden and repaints it.
showFast	Marks the control as visible.
sizeWindow	Changes the size, visibility, and Z order of the dialog.
tabStop	Add or remove the tab stop style for the control.
textSize	Computes the width and height in pixels of the specified string of text when displayed by this control.
title	Gets the text of the control.
title =	Sets the text of the control.
transparentText	Sets the text drawing mode in a device context to transparent, (background is not changed.)
update	Invalidates the entire client area of the control.
updateWindow	Updates the client area of this window by sending a paint message to the window, if the window's update region is not empty.
useUnicode	Sets the format flag telling the control to use, or not use, Unicode.
useVersion	Informs the control that the programmer is expecting a behavior associated with a particular common control library version.
usingUnicode	Determines if the control is using Unicode or not.
usingVersion	Returns the version number for the control that was set by the most recent *useVersion*method call.
vScrollPos	Returns the position of the vertical scroll bar in the dialog control.
windowRect	Returns a **`Rect`** object containng the dimensions of the control in pixels.
write	Writes text in a dialog control using the given font, style, and color at the position specified.
writeDirect	Writes text in a dialog control at the position specified using a device context.

9.2. Attribute Methods

This section describes the attributes of the dialog control object.

9.2.1. factorX (Attribute)

```
WindowBase::factorX


>>--factorX-------------------------------------><

>>--factorX = ratio-----------------------------><
```

9.2.2. factorY (Attribute)

```
WindowBase::factorY


>>--factorY--------------------------------------><

>>--factorY = ratio------------------------------><
```

9.2.3. hDlg (Attribute)

```
>>--hDlg-----------------------------------------><

>>--hDlg = varName-------------------------------><
```

The *hDlg* attribute reflects the window *handle* of the *underlying* Windows dialog that this dialog control belongs to.

hDlg get:
: Returns the window handle of the underlying dialog that this dialog control belongs to.

hDlg set:
: The programmer can not set the *hDlg* attribute. It is set correctly by the ooDialog framework and can not be changed.

9.2.4. hwnd

```
WindowBase::hwnd


>>--hwnd-----------------------------------------><
```

9.2.5. id (Attribute)

```
>>--id-------------------------------------------><

>>--id = varName---------------------------------><
```

The *id* attribute reflects the numeric resource ID for this dialog control.

id get:
: Returns the resource ID of the dialog control. This is always the whole number resource ID, not the *symbolic*) ID, even if symbolic IDs are used for the control.

id set:
: The programmer can not set the value of this attribute. Its value is the value of the *underlying* dialog control's ID, which is set when the dialog control is created by the operating system and can not be changed.

9.2.6. initCode (Attribute)

```
WindowBase::initCode


>>--initCode------------------------------------><

>>--initCode = code-----------------------------><
```

9.2.7. oDlg (Attribute)

```
>>--oDlg----------------------------------------><

>>--oDlg = varName------------------------------><
```

The *oDlg* attribute reflects the Rexx dialog object that this dialog control belongs to.

oDlg get:
Returns the Rexx dialog object to which this dialog control belongs.

oDlg set:
The programmer can not set the *oDlg* attribute. It is set correctly by the ooDialog framework and can not be changed.

9.2.8. sizeX (Attribute)

```
WindowBase::sizeX


>>--sizeX---------------------------------------><

>>--sizeX = dialogUnits-------------------------><
```

9.2.9. sizeY (Attribute)

```
WindowBase::sizeY


>>--sizeY---------------------------------------><

>>--sizeY = dialogUnits-------------------------><
```

9.2.10. pixelCX (Attribute)

```
WindowBase::pixelCX


>>--pixelCX-------------------------------------><
```

9.2.11. pixelCY (Attribute)

```
WindowBase::pixelCY


>>--pixelCY--------------------------------------><
```

9.3. Basic Window Methods

Recall that in the Windows graphical user interface, everything is a window. The *WindowBase* class is a mixin class that contains methods common to all windows. Since a dialog control is a window, it inherits the *WindowBase* class, the same as the *dialog* object does. All *WindowBase* methods are common to both dialogs and dialog controls. This section lists all the basic window methods for the dialog control object.

9.3.1. childWindowFromPoint

```
WindowBase::childWindowFromPoint


>>--childWindowFromPoint(--point--+----------+--)------------------------------><
                                  +-,-flags--+
```

9.3.2. clear

```
WindowBase::clear


>>--clear----------------------------------------><
```

9.3.3. client2screen

```
WindowBase::client2screen


>>--client2screen(--pointOrRect--)---------------><
```

9.3.4. clientRect

```
WindowBase::clientRect


>>--clientRect(--+--------+--)-------------------><
                 +--hwnd--+
```

9.3.5. clientToScreen

```
WindowBase::clientToScreen
```

```
>>--clientToScreen(--x--,--y--)------------------><
```

9.3.6. disable

```
WindowBase::disable


>>--disable--------------------------------------><
```

9.3.7. display

```
WindowBase::display
```

9.3.8. draw

```
WindowBase::draw


>>--draw-----------------------------------------><
```

9.3.9. enable

```
WindowBase::enable


>>--enable---------------------------------------><
```

9.3.10. foregroundWindow

```
WindowBase::foregroundWindow


>>--foregroundWindow-----------------------------><
```

9.3.11. getClientRect

```
WindowBase::getClientRect


>>--getClientRect(--+------+--)------------------><
                    +-hwnd-+
```

9.3.12. getExStyleRaw

```
WindowBase::getExStyleRaw


>>--getExStyleRaw--------------------------------><
```

9.3.13. getID

```
WindowBase::getID


>>--getID----------------------------------------><
```

9.3.14. getPos

```
WindowBase::getPos


>>--getPos---------------------------------------><
```

9.3.15. getRealPos

```
WindowBase::getSize


>>--getRealPos-----------------------------------><
```

9.3.16. getRealSize

```
WindowBase::getRealSize


>>--getRealSize----------------------------------><
```

9.3.17. getRect

```
WindowBase::getRect


>>--getRect--------------------------------------><
```

9.3.18. getSize

```
WindowBase::getSize


>>--getSize--------------------------------------><
```

9.3.19. getStyleRaw

```
WindowBase::getStyleRaw


>>--getStyleRaw----------------------------------><
```

9.3.20. getText

```
WindowBase::getText


>>--getText--------------------------------------><
```

9.3.21. getTextSizePx

```
WindowBase::getTextSizePx


>>--getTextSizePx(-text--)-----------------------><
```

9.3.22. getTextSizeScreen

```
WindowBase::getTextSizeScreen


>>--getTextSizeScreen(-text--+---------+--+------------+--+-------------+-)----><
                             +-,-type--+  +-,-fontSrc--+  +-,-fontSize--+
```

9.3.23. hide

```
WindowBase::hide


>>--hide-----------------------------------------><
```

9.3.24. hideFast

```
WindowBase::hideFast


>>--hideFast-------------------------------------><
```

9.3.25. isEnabled

```
WindowBase::isEnabled


>>--isEnabled------------------------------------><
```

9.3.26. isVisible

```
WindowBase::isVisible


>>--isVisible------------------------------------><
```

9.3.27. mapWindowPoints

```
WindowBase::mapWindowPoints


>>--mapWindowPoints(--hwndTo--,--points--)------><
```

9.3.28. move

```
WindowBase::move


>>--move(--xPos--,--yPos--+------------+--)-----><
                          +-,-showOpts--+
```

9.3.29. moveTo

```
WindowBase::moveTo


Form 1:

>>--moveTo(--point--+--------------+--)---------><
                    +--,-showOpts--+
Form 2:

>>--moveTo(--x,--y--+--------------+--)---------><
                    +--,-showOpts--+
Generic form:

>>--moveTo(--newPos--+--------------+--)--------><
                     +--,-showOpts--+
```

9.3.30. moveWindow

```
WindowBase::moveWindow


Form 1:

>>--moveWindow(--hwndBehind--,--point--+--------------+--)------><
                                       +--,-showOpts--+
Form 2:

>>--moveWindow(--hwndBehind--,--x,--y--+--------------+--)------><
                                       +--,-showOpts--+
Generic form:

>>--moveWindow(--hwndBehind--,--newPos--+--------------+--)-----><
                                        +--,-showOpts--+
```

9.3.31. redraw

```
WindowBase::redraw
```

```
>>--redraw---------------------------------------><
```

9.3.32. redrawClient

```
WindowBase::redrawClient


>>--redrawClient(--+------------+--)-------------><
                   +--eraseBkg--+
```

9.3.33. resize

```
WindowBase::resize


>>--resize(--width--,--height--+-------------+--)---------------><
                               +-,-showOpts--+
```

9.3.34. resizeTo

```
WindowBase::resizeTo


Form 1:

>>--resizeTo(--size--)---------------------------><

Form 2:

>>--resizeTo(--cx,--cy--)------------------------><

Generic form:

>>--resizeTo(--newSize--)------------------------><
```

9.3.35. screen2client

```
WindowBase::screen2client


>>--screen2client(--pointOrRect--)---------------><
```

9.3.36. screenToClient

```
WindowBase::screenToClient


>>--screenToClient(--x--,--y--)------------------><
```

9.3.37. sendMessage

```
WindowBase::sendMessage
```

```
>>--sendMessage(--id--,--msg--,--wParam--,--lParam--)----------><
```

9.3.38. sendMessageHandle

```
sendMessageHandle


>>--sendMessageHandle(--id--,--msg--,--wParam--,--lParam--)-----><
```

9.3.39. setRect

```
WindowBase::setRect


Form 1:

>>--setRect(--rectangle--+-----------+--)-------><
                         +-,-showOpts-+

Form 2:

>>--setRect(--point--,--size--+-----------+--)----------------><
                              +-,-showOpts-+

Form 3:

>>--setRect(--x-,--y-,--cx-,--cy--+-----------+--)------------><
                                  +-,-showOpts-+

Generic form:

>>--setRect(--ptSizeRectangle--+-----------+--)---------------><
                               +-,-showOpts-+
```

9.3.40. setStyleRaw

```
WindowBase::setStyleRaw


>>--setStyleRaw(--value--)----------------------><
```

9.3.41. setText

```
WindowBase::setText


>>--setText(--newText--)------------------------><
```

9.3.42. setTitle

```
WindowBase::setTitle
```

```
>>--setTitle(--newText--)------------------------><
```

9.3.43. setWindowPos

```
WindowBase::setWindowPos


Form 1:

>>--setWindowPos(--hwndBehind--,--rectangle--+-----------+--)----------------><
                                             +-,-showOpts-+

Form 2:

>>--setWindowPos(--hwndBehind--,--point--,--size--+-----------+--)-----------><
                                                  +-,-showOpts-+

Form 3:

>>--setWindowPos(--hwndBehind--,--x-,--y-,--cx-,--cy--+-----------+--)-------><
                                                      +-,-showOpts-+

Generic form:

>>--setWindowPos(--hwndBehind--,--ptSizeRectangle--+-----------+--)----------><
                                                   +-,-showOpts-+
```

9.3.44. show

```
WindowBase::show


>>--show-----------------------------------------><
```

9.3.45. showFast

```
WindowBase::showFast


>>--showFast-------------------------------------><
```

9.3.46. sizeWindow

```
WindowBase::sizeWindow


Form 1:

>>--sizeWindow(--hwndBehind--,--size--+-------------+--)-------><
                                      +--,-showOpts--+

Form 2:

>>--sizeWindow(--hwndBehind--,--cx,--cy--+-------------+--)----><
```

```
                                         +--,-showOpts--+

Generic form:

>>--sizeWindow(--hwndBehind--,--newSize--+--------------+--)----><
                                         +--,-showOpts--+
```

9.3.47. title

```
WindowBase::title


>>--title----------------------------------------><
```

9.3.48. title=

```
WindowBase::title=


>>--title=newText--------------------------------><
```

9.3.49. update

```
WindowBase::update


>>--update---------------------------------------><
```

9.3.50. updateWindow

```
WindowBase::updateWindow


>>--updateWindow---------------------------------><
```

9.3.51. windowRect

```
WindowBase::windowRect


>>--windowRect(--+--------+--)-------------------><
                 +--hwnd--+
```

9.4. Extended Window Methods

The methods implemented by *WindowsExtensions* class are listed in this section. The class name, *WindowExtensions* would seem to imply that the methods were common to all windows. However, the methods of this class are really just extensions to the original ooDialog framework, and many of the methods are not window specific methods.

In many ways a lot of these methods do not make sense in the context of a dialog control. However, the methods have been dialog control methods for many years and must therefore remain dialog control methods for the sake of backwards compatibility. In almost all instances, the methods are best used in the context of a *dialog* object rather than a dialog control object.

9.4.1. createBrush

```
WindowExtensions::createBrush


>>--createBrush(--+---------+--+------------------+--)-------------------------><
                  +--color--+  +-,-brushSpecifier-+
```

9.4.2. createFont

```
WindowExtensions::createFont


>>--createFont(--+----------+-+------------+-+---------+-+-------------+--)----><
                 +-fontName-+ +-,-fontSize-+ +-,-style-+ +-,-fontWidth-+
```

9.4.3. createFontEx

```
WindowExtensions::createFontEx


>>--createFontEx(--fontName-+--------------+--+---------------+--)-------------><
                            +-,-pointSize--+  +-,-additional--+
```

9.4.4. createPen

```
WindowExtensions::createPen


>>--createPen(--+-------+--+----------+--+----------+--)-----------------------><
                +-width-+  +-,-style--+  +-,-color--+
```

9.4.5. deleteFont

```
WindowExtensions::deleteFont


>>--deleteFont(--hFont--)------------------------><
```

9.4.6. deleteObject

```
WindowExtensions::deleteObject


>>--deleteObject(--obj--)------------------------><
```

9.4.7. drawAngleArc

```
WindowExtensions::drawAngleArc


>>--drawAngleArc(-dc-,-xs-,-ys-,-x-,-y-,-radius-,-startangle-,-sweepangle-)----><
```

9.4.8. drawArc

```
WindowExtensions::drawArc


>>--drawArc(--dc-,-x-,-y-,-x2-,-y2--+-------+--+-------+--+-------+--+-------+--)---><
                                    +-,-r1x-+  +-,-r1y-+  +-,-r2x-+  +-,-r2y-+
```

9.4.9. drawLine

```
WindowExtensions::drawLine


>>--drawLine(--dc--,--+-------+--,--+-------+--,--toX--,--toY--)-><
                      +-fromX-+     +-fromY-+
```

9.4.10. drawPie

```
WindowExtensions::drawPie


>>--drawPie(--dc-,-x-,-y-,-x2-,-y2--+-------+--+-------+--+-------+--+-------+--)---><
                                    +-,-r1x-+  +-,-r1y-+  +-,-r2x-+  +-,-r2y-+
```

9.4.11. drawPixel

```
WindowExtensions::drawPixel


>>--drawPixel(--dc--,--x--,--y--,--color--)------><
```

9.4.12. fillDrawing

```
WindowExtensions::fillDrawing


>>--fillDrawing(--dc--,--x--,--y--,--color--)----><
```

9.4.13. fillRect

```
WindowExtensions::fillRect
```

```
>>--fillRect(--dc--,--rect--,--brush--)----------><
```

9.4.14. freeDC

```
WindowExtensions::freeDC

>>--freeDC(--dc--)-------------------------------><
```

9.4.15. fontColor

```
WindowExtensions::fontColor

>>--fontColor(--color--,--dc--)------------------><
```

9.4.16. fontToDC

```
WindowExtensions::fontToDC

>>--fontToDC(--dc--,--hFont--)-------------------><
```

9.4.17. getArcDirection

```
WindowExtensions::getArcDirection

>>--getArcDirection(--dc--)----------------------><
```

9.4.18. getDC

```
WindowExtensions::getDC

>>--getDC----------------------------------------><
```

9.4.19. getFont

```
WindowExtensions::getFont

>>--getFont--------------------------------------><
```

9.4.20. getPixel

```
WindowExtensions::getPixel
```

```
>>--getPixel(--dc--,--x--,--y--)-----------------><
```

9.4.21. getSysBrush

```
WindowExtensions::getSysBrush


>>--getSysBrush(--sysColor--)--------------------><
```

9.4.22. getTextAlign

```
WindowExtensions::getTextAlign


>>--getTextAlign(--hDC--)------------------------><
```

9.4.23. getTextExtent

```
WindowExtensions::getTextExtent


>>--getTextExtent(--hDC--,--text--)--------------><
```

9.4.24. hScrollPos

```
WindowExtensions::hScrollPos


>>--hScrollPos-----------------------------------><
```

9.4.25. loadBitmap

```
WindowExtensions::loadBitmap


>>--loadBitmap(--bmpFilename--+------------+--)--><
                              +-,-loadOpt--+
```

9.4.26. objectToDC

```
WindowExtensions::objectToDC


>>--objectToDC(--dc--,--obj--)-------------------><
```

9.4.27. opaqueText

```
WindowExtensions::opaqueText
```

```
>>--opaqueText(--dc--)---------------------------><
```

9.4.28. rectangle

```
WindowExtensions::rectangle


>>--rectangle(--dc--,--x--,--y--,--x2--,--y2--+----------+--)--><
                                              +-,-keyWord-+
```

9.4.29. removeBitmap

```
WindowExtensions::removeBitmap


>>--removeBitmap(--hBitmap--)--------------------><
```

9.4.30. scroll

```
WindowExtensions::scroll


>>--scroll(--cx--,--cy--)------------------------><
```

9.4.31. setArcDirection

```
WindowExtensions::setArcDirection


>>--setArcDirection(--dc--+-------------+--)----><
                          +-,-direction--+
```

9.4.32. setFont

```
WindowExtensions::setFont


>>--setFont(--fontHandle--+---------+--)----------><
                          +-,redraw-+
```

9.4.33. setHScrollPos

```
WindowExtensions::setHScrollPos


>>--setHScrollPos(--position--+----------+--)---><
                              +-,-redraw--+
```

9.4.34. setTextAlign

```
WindowExtensions::setTextAlign


>>--setTextAlign(--hDC--,--align--)--------------><
```

9.4.35. setVScrollPos

```
WindowExtensions::setVScrollPos


>>--setVScrollPos(--position--+-----------+--)---><
                              +-,-redraw--+
```

9.4.36. transparentText

```
WindowExtensions::transparentText


>>--transparentText(--dc--)----------------------><
```

9.4.37. vScrollPos

```
WindowExtensions::vScrollPos


>>--vScrollPos-----------------------------------><
```

9.4.38. write

```
WindowExtensions::write


>>--write(-x-,-y-,-text-+---------+-+---------+-+--------+-+------+-+------+-)-><
                        +-,-fName-+ +-,-fSize-+ +-,-opts-+ +-,-fg-+ +-,-bk-+
```

9.4.39. writeDirect

```
WindowExtensions::writeDirect


>>--writeDirect(--dc--,--xPos--,--yPos--,--text--)--------------><
```

9.5. assignFocus

```
>>--assignFocus----------------------------------><
```

Assigns the input focus to this dialog control.

Arguments:

This method has no arguments.

Return value:

Returns 0, always.

Remarks:

When a dialog control is assigned the focus, the operating system does more than just set the input focus to the window. The Windows dialog manager also updates the default push button border, sets the default control identifier, and automatically selects the text of an edit control (if the target window is an edit control).

9.6. clearRect

```
Form 1:

>>--clearRect(--rectangle--)---------------------><


Form 2:

>>--clearRect(--pt1--,--pt2--)-------------------><


Form 3:

>>--clearRect(--x-,--y-,--cx-,--cy--)------------><


Generic form:

>>--clearRect(--rectCoordinates--)---------------><
```

The *clearRect* method clears the specified rectangular within the client area of this control. The rectangle is *cleared* by redrawing it with the background brush set to the typical background color for dialog boxes and three dimensional elements.

Arguments:

The arguments are:

rectCoordinates [required]

The coordinates of the rectangle to be cleared. The coordinates specify the upper left and lower right corners of the rectangle. The corners can be specified using either a *Rect* object, two *Point* objects, or the individual x and y coordinates of each corner.

The coordinates are specified as *client area* coordinates, in pixels.

Return value:

0 on success, 1 on error.

Remarks:

The rectangle is not really cleared, but rather is redrawn using the normal background color for a dialog box.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

9.7. connectCharEvent

```
>>--connectCharEvent(--+--------------+--)-------><
                       +--methodName--+
```

The *connectCharEvent* method connects a keyboard *character* event notification to this dialog control with a method in the Rexx dialog.

Arguments:

The single argument is:

methodName [optional]

The name of the Rexx method to be connected to the character event. If this argument is omitted, the ooDialog framework uses a method name of: *onChar*.

Return value:

Returns **`.true`** on success and **`.false`** on error.

Remarks:

The operating system translates the numeric *VK* key code produced by a key down / key up combination into a numeric *character* code. The character code is sent to the window with the keyboard focus in a character event notification. For enhanced 101-key and 102-key keyboards, the *CHAR* event handler will also receive extended key notifications. The extended keys are the INS, DEL, HOME, END, PAGE UP, PAGE DOWN and the arrow keys in the clusters to the left of the numeric keypad.

There is not a one to one correspondence between the numeric value of a virtual key code and the numeric character code for the same key. For example the character key code for the dash - is 45, but 45 is the virtual key code for INSERT. I.e., **`45 == .VK~INSERT`** and **`45~d2c == -`**.

The character codes expected to reach the *character event handler* are in the following table. The explanation for the character event handler describes how to determine which character is received.

Table 9.2. Numeric Character Code for the Character Event

ASCII Characters	Numeric Codes
Ctrl-A through Ctrl-Z	1 through 26
Ctrl-[	27
Ctrl-]	29
Normal alphabetic characters	use **d2c**
Punctuation characters	use **d2c**
Numeric characters	use **d2c**
Extended keys use the *VK* class	

Note that the Microsoft documentation states that character notifications are only generated for keys that are mapped to ASCII characters by the keyboard driver. This implies that different keyboards may produce different character codes than those listed above. The list was produced through experimentation by the author, using an extended keyboard, and is in general correct. Some specific details may be different for other keyboards.

Details

Raises syntax errors when incorrect usage is detected.

This method requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version* method to determine the current version of the library.

Example:

This is a complete working example. It can be used to experiment with the *connectCharEvent* method. As the user types in the edit box, information from the character event are printed to the console. This allows the user to see what key presses reach the application as characters:

```
/* Print characters received */

  .application~setDefaults('O', , .false)
  .constDir[IDC_EDIT] = 200

  dlg = .TestDialog~new
  dlg~execute("SHOWTOP", IDI_DLG_OOREXX)

::requires "ooDialog.cls"

::class 'TestDialog' subclass UserDialog

::method init
  forward class (super) continue
  self~create(30, 30, 250, 200, "CHAR Tester", "CENTER")

::method defineDialog

  self~createEdit(IDC_EDIT, 10, 10, 230, 155, "MULTILINE")
  self~createPushButton(IDOK, 135, 175, 50, 14, "DEFAULT", "Ok")
  self~createPushButton(IDCANCEL, 190, 175, 50, 14, , "Cancel")

::method initDialog

  self~newEdit(IDC_EDIT)~connectCharEvent(onChar)

::method onChar unguarded
  use arg char, isShift, isCtrl, isAlt, misc, control

  if misc~pos('extended') <> 0 then c = .VK~key2name(char)
  else c = char~d2c

  say 'Char:' char c 'isShift' isShift 'isCtrl' isCtrl 'isAlt' isAlt 'misc' misc
  return .true
```

9.7.1. Char Event Handler

The *event* handler for the char event is invoked when the dialog control has the keyboard focus and the user types a key or a keyboard combination that the operating system translates to a *character*.

The programmer must return a value from the event handler and the interpreter waits for this return. Generic information on how to *code* event handlers is included in the documentation for the *EventNotification* class.

```
::method onChar unguarded
  use arg char, isShift, isCtrl, isAlt, misc, controlObj

  return response
```

Event Handler Method Arguments:

The event handling method receives 6 arguments:

char

The numeric character code of the key pressed, or on extended keyboards, the virtual key code of an extended key.

For enhanced 101-key and 102-key keyboards, the extended keys are the INS, DEL, HOME, END, PAGE UP, PAGE DOWN and the arrow keys in the clusters to the left of the numeric keypad.

To determine the character received use **`char~d2c`** when *isCtrl* is false and *misc* does *not* contain the word *extended*. Use the *VK* class when *isCtrl* is false and *misc* contains the word *extended*.

In addition, the characters Ctrl-A through Ctrl-Z, plus Ctrl-[and Ctrl-] are received. For this group of characters, the *isCtrl* argument will be true and the numeric value of *char* will be 1 through 26 for Ctrl-A through Ctrl-Z, 27 for Ctrl-[, and 29 for Ctrl-]

isShift

A boolean (true or false) that denotes whether a shift key was down or up at the time of the key press. It will be true if a shift key was down and false if the shift key was not down.

isCtrl

True if a control key was down at the time of the key press, false if it was not.

isAlt

True if an alt key was down at the time of the key press, false if it was not.

extraInfo

This argument is a string containing keywords. It supplies extra information about the keyboard state at the time of a key press event. The string will contain some combination of these keywords

extended

The character event is for one of the extended keys previously mentioned, INS, DEL, HOME, END, PAGE UP, PAGE DOWN, or one of the arrow keys.

numOn

Num Lock was on at the time of the key press event.

numOff

Num Lock was off.

capsOn

Caps Lock was on at the time of the key press event.

capsOff

Caps Lock was off.

scrollOn

Scroll Lock was on at the time of the key press event.

scrollOff

Scroll Lock was off.

lShift
: The left shift key was down at the time of the key press event.

rShift
: The right shift key was down.

lControl
: The left control key was down at the time of the key press event.

rControl
: The right control key was down.

lAlt
: The left alt key was down at the time of the key press event.

rAlt
: The right alt key was down.

controlObj
: The dialog control object representing the underlying control that received the keyboard character event notification.

Return:

The event handler must return **`.true`**, **`.false`**, or a numeric character code. Returning **`.true`** passes the event notification on to the control unchanged. Returning **`.false`** prevents the character from being sent on to the control. This has the effect of *swallowing* the character. The control will not be aware of the character. Passing a numeric character back has the effect of replacing what the character was, with the character passed back.

Example

This example comes from some code my nephew wrote. I wrote a typing tutorial for my niece and nephews, they each had their own copy. Each week their typing scores were expected to get better. If their score was worse, they were assigned extra chores for the next week. My nephew injected the following code into his brother's typing tutorial. He thought it was hilarious, his brother was not amused:

```
  self~newEdit(IDC_EDIT_TYPING_WINDOW)~connectCharEvent(onChar)

::method onChar unguarded
  use arg char, isShift, isCtrl, isAlt, misc, control

  if misc~pos('extended') <> 0 then do
    if char == .VK~END then char = .VK~HOME
    else if char == .VK~HOME then char = .VK~END
    else if char == .VK~LEFT then char = .VK~RIGHT
  end
  else do
    if char~d2c == 'a' then char = 'z'~c2d
    else if char~d2c == 'i' then char = 'k'~c2d
  end

  return char
```

9.8. connectFKeyPress

```
>>--connectFKeyPress(--methodName--)-------------><
```

The *connectFKeyPress* method connects all F Key key press events to a method in the Rexx dialog.

Arguments:

The single arguments is:

methodName [required]

The name of the method that is to be invoked when a F Key key press event happens. The method name can not be the empty string. The method is defined by the programmer in the dialog class that is the parent to the dialog control.

Return value:

The possible return values are:

0

Success.

-2

The underlying mechanism in the Windows API that is used to capture key events failed.

-6

The maximum number of connections has been reached.

-7

The *methodName* method is already connected to a key down event for this dialog.

Remarks:

The *connectFKeyPress* method is a convenience method for the *connectKeyPress* method. It is equivalent to using the *connectKeyPress* method and supplying the FKEYS keyword for the *keys* argument and omitting the *filter* argument. The *connectKeyPress* method documentation goes into deeper detail in explaining how the *connectKeyPress* method, and by extension the *connectFKeyPress* method, works.

Details:

This method requires CommonControl*Library* version 6.0 or later. If necessary use the *comCtl32Version* method to determine the current version of the library.

Raises syntax errors when some incorrect usage is detected.

Example:

The following example connects the F key down events for an edit control. The application determines which F key was pressed and takes appropriate action. For this application, the F3 key is the search key.

```
::method initDialog
  expose editControl lastSearchToken

  editControl = self~newEdit(IDC_EDIT)
  editControl~connectFKeyPress(onFKey)
  ...
  lastSearchToken = ""
  ...

::method onFKey
  expose editControl lastSearchToken
  use arg key

  select
    when key == .VK~F2 then do
```

```
      ...
    end
    when key == .VK~F3 then do
      description = "What word would you like to search for?"
      lastSearchToken = inputBox(description, "Search", lastSearchToken)

      if lastSearchToken <> "" then
        self~findAndHighlight(lastSearchToken, editControl)
    end
    when key == .VK~F4 then do
      ...
    end
    ...
  end
  ...
```

9.9. connectKeyPress

```
>>--connectKeyPress(--methodName--,--keys--+----------+--)-----><
                                           +-,-filter--+
```

The *connectKeyPress* method connects a key press *event* notification with a method in the Rexx dialog. A single key or multiple keys can be connected to the same method. Multiple methods can be connected for key press events, but only 1 method can be connected to any single key.

Arguments:

The arguments are:

methodName [required]

The name of the method that is to be invoked when the key press event happens. The method name can not be the empty string. The method is defined by the programmer in the dialog class that is the parent to the dialog control.

keys [required]

The key (or keys) for which the key press event is to be connected. A single key or multiple keys can be specified. A range of keys can be used. Each single key or range of keys is separated by a comma. A range of keys is denoted by using the dash character "-". White space within the *keys* argument is ignored. This argument can not be the empty string.

The keys are specified by the numeric value defined by Microsoft for its virtual key set. These numeric values are 0 through 255. There are some integer values between 0 and 255 that do not have a virtual key assigned to them. For example, 0, 7, 10, 11, and 255 are not used. The *VK* class contains constants for all of the defined virtual keys.

In addition, there are a few keywords that can be used to specify some common key ranges. These keywords are:

ALL

All keys.

FKEYS

All Function keys, other than F1. (In Windows the F1 key is the help key and the *connectHelp* method should be used for F1.)

ALPHA

The keys A though Z.

NUMERIC

The keys 0 through 9. Note that these are the normal number keys, not the keypad numbers on an enhanced keyboard.

ALPHANUMERIC

The keys A through Z and 0 through 9.

Note that case is insignificant for these keywords as is the order of the keywords. A keyword not in the list will result in a return of -9. However, if the argument contains other valid key tokens, those keys will be connected to the method. If there are no other valid key tokens, then no connection is made.

filter [optional]

A (simplistic) filter that is applied to the key press event for the key(s) specified. The filter is a string of keywords separated by blanks. (Case is not significant, neither is the order of the words. Any words other than the specified keywords are ignored.) The possible keywords are: **`SHIFT, CONTROL, ALT, AND, NONE, VIRTUAL.`**

The VIRTUAL keyword can be abbreviated to VIRT if desired. The VIRTUAL keyword effects how the test for the shift, control, and alt key is performed. By default the physical state of the keyboard is checked to see if the control, alt, or shift key is depressed. However, it is common in Windows to use keystroke programs that inject keystrokes into other application windows. Testing the physical state of the keyboard will not detect combination keystrokes like Ctrl-S, Alt-L, etc., that are inserted by keystroke programs because the physical state of the modifier keys control and alt will not be depressed. If the VIRTUAL keyword is used, the test for the modifier keys being down will be altered in a way that will detect if the virtual state of the key is down. This test will detect key events inserted into the Rexx application by third party keystroke programs.

Shift, control, and alt specify that the corresponding key must be down at the time of the key press event. These keywords are combined in a boolean expression. The default is an OR expression. If the AND keyword is present then the boolean expression is an AND expression. If the NONE keyword is used, it means that none of the shift, control, or alt keys can be down at the time of the key press event. (When NONE is used, all other words, except VIRTUAL, in the string are ignored.)

Some examples may make this more clear:

```
::method initDialog

  editControl = self~newEdit(IDC_EDIT1)

  -- Using the below, the onAltCD method would be invoked when the user types
  -- Alt-Shift-C or Alt-Shift-D when the edit control has the focus.  But the
  -- method would not be invoked for Alt-C or Shift-D (or any other key press
  -- event.)

  keys = .VK~C "," .VK~D
  editControl~connectKeyPress(onAltCD, keys, "ALT AND SHIFT")

  -- The below would invoke the onAltCD method any time a C or a D was typed,
  -- when the edit control has the focus, with either the Alt or the Control key
  -- also being down.  This would include Alt-C, Alt-Shift-C, Ctrl-Alt-Shift-C,
  -- etc..

  editControl~connectKeyPress(onAltCD, keys, "ALT CONTROL")

  -- The below would invoke the onAltCD method only when Alt-C or Alt-D was
  -- typed.
```

```
editControl~connectKeyPress(onAltCD, keys, "ALT AND")

-- The below would invoke the onF4 method when the edit control has the focus,
-- only when the F4 key was pressed by itself. Alt-F4, Ctrl-F4, etc., would
-- not invoke the method.

editControl~connectKeyPress(onF4, .VK~F4, "NONE")
```

Return value:

The possible return values are:

0

Success.

-2

The underlying mechanism in the Windows API that is used to capture key events failed.

-6

The maximum number of connections has been reached.

-7

The *methodName* method is already connected to a key down event for this dialog.

-8

The *filter* argument is not correct.

-9

An incorrect format for the *keys* argument. Note that it is possible to get a return of -9 but still have some keys connected. For instance in the following example the C and D keys would be connected and the filter applied. The ""dog"" token would result in -9 being returned:

```
editControl = self~newEdit(IDC_EDIT_ENTRY)

keys = .VK~C ", dog," .VK~D
ret = editControl~connectKeyPress('onAltCD', keys, "ALT AND SHIFT")
say 'Got a return of:' ret
say "Have connection to onAltCD?" self~hasKeyPressConnection('onAltCD')

-- The output would be:
Got a return of: -9
Have connection to onAltCD? 1
```

Remarks:

There is a maximum limit of 63 methods, per dialog, that can be connected to key press events. Connections can be removed using the *disconnectKeyPress* method if there is no longer a need for a notification of a key press.

The dialog object also has a *connectKeyPress* method. It is important to note this distinction between the two methods. The method of the dialog control object (this method) will capture any key press event when the specific dialog control has the focus. The method of the dialog object will capture all key press events when the dialog is the active window.

This includes key presses when a dialog control in the dialog has the focus. This implies that if you connect the same key press event to both the dialog and to a specific dialog control, if the key press event occurs when the dialog control has the focus, you will receive two event notifications.

Details:

This method requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version* method to determine the current version of the library.

Raises syntax errors when some incorrect usage is detected.

Event Handler Method Arguments:

The ooDialog method connected to the key press event will receive the following five arguments in the order listed:

keyCode
: The numeric code of the key pressed.

shift
: A boolean (true or false) that denotes whether a shift key was down or up at the time of the key press. It will be true if a shift key was down and false if the shift key was not down.

control
: True if a control key was down at the time of the key press, false if it was not.

alt
: True if an alt key was down at the time of the key press, false if it was not.

extraInfo
: This argument is a string containing keywords. It supplies extra information about the keyboard state at the time of a key press event. The string will contain some combination of these keywords

  numOn
  : Num Lock was on at the time of the key press event.

  numOff
  : Num Lock was off.

  capsOn
  : Caps Lock was on at the time of the key press event.

  capsOff
  : Caps Lock was off.

  scrollOn
  : Scroll Lock was on at the time of the key press event.

  scrollOff
  : Scroll Lock was off.

  lShift
  : The left shift key was down at the time of the key press event.

  rShift
  : The right shift key was down.

  lControl
  : The left control key was down at the time of the key press event.

  rControl
  : The right control key was down.

lAlt
: The left alt key was down at the time of the key press event.

rAlt
: The right alt key was down.

Example:

The following example could be from an application that supplies extended editing abilities for a multi-line edit control. When the edit control has the focus the user can use Alt-D to delete the current word, Ctrl-D to delete the current line, and Shift-Alt-D delete the current paragraph:

```
::method initDialog
  expose editControl

  editControl = self~newEdit(IDC_EDIT)
  ...

  -- Capture the D key press when either the Alt or Control keys are
  -- also pressed.
  editControl~connectKeyPress(onDPress, .VK~D, "ALT CONTROL")
  ...

::method onDPress
  expose editControl
  use arg key, shift, control, alt, info

  -- Determine which of the key press combinations this is and take
  -- appropriate action if it is a combination we are interested in.

  isAltD = \ shift & \ control & alt
  isCtrlD = \ shift & control & \ alt
  isShiftAltD = shift & \ control & alt

  if isAltD then self~deleteWord(editControl)
  if isCtrlD the self~deleteLine(editControl)
  if isShiftAltD then self~deleteParagraph(editControl)
```

9.10. data

```
>>--data-----------------------------------------><
```

The *data* method retrieves the current *data* of the dialog control.

Arguments:

This method does not take any arguments.

Return value:

The current *data* of the dialog control. Exactly what the format of the *data* takes is dependent on the type of the control. The Data Attribute Methods *section* of this reference has a detailed discussion on data attributes and the meaning of the *data* of a dialog control.

Remarks:

This method was really intended to be used on controls that have previously been connected to a data attribute. But, since the dialog control object, itself, knows which type of control it is, the method works whether the control is connected or not. However, the abstraction of viewing a dialog control's state as data, represented by a single string, is restrictive and somewhat out of date. It is more flexible, and often easier, to use the methods of the specific dialog control object to determine the control's state.

Example:

The following example determines if the restart check box is checked, and if so sets a flag that tells the application to restart the computer:

```
::method leaving
  expose restartChk fancyFont logoBmp appCleanUpParams

  if restartChk~data == 1 then do
    self~deleteFont(fancyFont)
    logoBmp~release

    appCleanUpParams['NoErrs']  = .true
    appCleanUpParams['UserOK']  = .true
    appCleanUpParams['Restart'] = .true
  end
  else do
    ...
    appCleanUpParams['Restart'] = .false
  end
```

The same thing can be accomplished in this manner:

```
::method leaving
  expose restartChk fancyFont logoBmp appCleanUpParams

  if restartChk~checked then do
    self~deleteFont(fancyFont)
    logoBmp~release

    appCleanUpParams['NoErrs']  = .true
    appCleanUpParams['UserOK']  = .true
    appCleanUpParams['Restart'] = .true
  end
  else do
    ...
    appCleanUpParams['Restart'] = .false
  end
```

9.11. data=

```
>>--data = newData-------------------------------><
```

Sets the *data* of the dialog control to the new value specified.

Arguments:

newData [required]

The value used to set the *data* of the dialog control. The exact format of this value is dependent on the type of the control. The Data Attribute Methods *section* of this reference has a detailed discussion on data attributes and the meaning of the *data* of a dialog control.

Remarks:

This method was really intended to be used on controls that have previously been connected to a data attribute. But, since the dialog control object, itself, knows which type of control it is, the method works whether the control is connected or not. However, the abstraction of viewing a dialog control's state as data is restrictive and somewhat out of date. It is more flexible, and often easier, to use the methods of the specific dialog control object to set the control's state.

Example:

The following example checks the restart check box and unchecks the verify check box:

```
::method setRestart private
  expose restartChk verifyChk

  restartChk~data = 1
  verifyChk~data = 0
```

The same thing can be accomplished in this manner:

```
::method setRestart private
  expose restartChk verifyChk

  restartChk~check
  verifyChk~uncheck
```

9.12. disconnectKeyPress

```
>>--disconnectKeyPress(--+--------------+--)-----><
                         +--methodName--+
```

Disconnects a method in the Rexx dialog from a previously connected key press *event*.

Arguments:

The single argument is:

methodName [optional]

The method to be disconnected. If omitted then all key press events are disconnected.

Return value:

The possible return values are:

0

Success.

-2

While trying to disconnect the method, the underlying mechanism in the Windows API that is used to capture key events had an error. This is unlikely to happen.

-7

Either the *methodName* method is already disconnected, or there are no methods connected at all.

Remarks:

The dialog object also has a *disconnectKeyPress* method. The method of the dialog control object (this method) can only disconnect key press events that were set with the dialog control object's versions of *connectKeyPress* and *connectFKeyPress* methods. This method can not disconnect key press events that were set with the dialog object's versions of *connectKeyPress* and *connectFKeyPress* methods.

Details

This method requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version* method to determine the current version of the library.

Raises syntax errors when some incorrect usage is detected.

Example:

The following example could come from a dialog where the user can enable or disable the use of hot keys when she is working within an edit control. When the user presses the disable hot keys button, the dialog disables the hot keys for the edit control by removing the key press connections.

```
::method defineDialog

  ...
  self~createPushButton(IDC_PB_DISABLE, 60, 135, 65, 15, , "Disable Hot Keys", onDisable)
  ...

method onDisable
  editControl = self~newEdit(IDC_EDIT)
  editControl~disconnectKeyPress
```

9.13. getTextSizeDlg

```
>>--getTextSizeDlg(--text--+-------------+--+-------------+--+---------+--)----><
                           +-,-fontname--+  +-,-fontSize--+  +-,-hwnd--+
```

Calculates the size, (width and height,) in *dialog unit* for a given string.

Note: The convoluted arguments to this method are needed to retain backwards compatibility with older versions of ooDialog. The recommendation is that the ooDialog programmer not use this method of the dialog control object.

In general, dialog units are only of value in laying out the dialog controls before the underlying dialog is created. Once the underlying dialog is created, it makes more sense to work with pixels. In addition, dialog units are tied directly to the actual font used by a specific dialog. Therefore, the optional arguments do not make much sense. Since the dialog object directly knows which font it is using, and the dialog control does not, it does not make much sense for this method to be a method of a dialog control in the first place.

The ooDialog programmer is therefore **strongly** encouraged to use the *getTextSizeDu* method of the *dialog* object.

Arguments:

The arguments are:

text [required]

The string whose size is desired. If none of the optional arguments are specified then the font of the dialog that owns the control is used to calculate the size.

However, there are some exceptions to this if the method is invoked *after* the underlying dialog is created.

1. If this method is invoked through a dialog control object, then the size is calculated using the dialog control font. This usage is deprecated and the ooDialog programmer is **strongly** encouraged to not use this method in this manner.

2. If the fourth, optional, hwnd argument is used, then the font of that window is used to calculate the size. Again, the ooDialog programmer is **strongly** encouraged to not use this method in this manner.

fontName [optional]
: The name of the font to use to calculate the size needed for the string. Use this argument when the string will be displayed in a font **different** than the dialog font.

fontSize [optional]
: The size of the font named by the fontName argument. If this argument is omitted then the default font size is used. (Currently the default size is 8.) This argument is ignored completely when the fontName argument is omitted.

hwnd [optional]
: Optional. A valid window handle. The font of this window is used to calculate the text size. This argument is always ignored when the fontName argument is specified. As per the notes above the ooDialog programmer is encouraged not to use this argument.

Return value:

The size needed for the string is returned in a *Size* object. The size is specified in dialog units.

9.14. group

```
>>--group(--+-----------+--)---------------------><
            +-wantStyle-+
```

Adds or removes the *group* style for this control.

Arguments:

The only argument is

wantStyle [optional]
: A boolean (.true or .false) to indicate whether the control should have or not have the group style. True (the default) indicates the control should have the group style and false indicates the control should not have the style.

Return value:

Negative values indicate the function failed, positive values indicate success.

Less than 0
: The value is the negated operating system error code.

Greater than 0
: The window style of the dialog control prior to adding or removing the group style.

Remarks:

The group style controls how the user can navigate through the dialog using the keyboard. For most dialogs this does not change while the dialog is executing. However, in some dialogs the programmer may want to change the navigation depending on the options the user selects.

The negative return value from this method is a hold over from a period prior to the introduction of the *.systemErrorCode*. This method now sets the **`.systemErrorCode`**, making it easier to use the negative return as a failure indication and use the **`.systemErrorCode`** value to look up the actual error.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

9.15. hasKeyPressConnection

```
>>--hasKeyPressConnection(--+--------------+--)--><
                            +--methodName--+
```

Queries if a specific method in the Rexx dialog has a connection to a key down event, or if any methods are connected to a key down event.

Arguments:

The single argument is:

methodName [optional]

If *methodName* is specified, then this method checks if there is a key press event connected to that method. When the argument is omitted, then the check is if any methods are connected to key press events.

Return value:

Returns **`.true`** if a key press event connection exists, otherwise **`.false`**.

Remarks:

The dialog object also has a *hasKeyPressConnection* method. The method of the dialog control object (this method) can only check for connections that were set with the dialog control object's versions of *connectKeyPress* and *connectFKeyPress* methods. This method can not check for connections that were set with the dialog object's versions of *connectKeyPress* and *connectFKeyPress* methods.

Example:

The following example is from an application where the user can enable the use of hot keys when an edit control has the focus, or not. The reset push button is used in the application to reset the state of the dialog. One of the things done when the state is reset to check or uncheck a check box that shows whether hot keys are currently enabled or not.

```

::method defineDialog

  ...
  self~createCheckBox(IDC_CHECK_FKEYSENABlED, 30, 60, , , , "Hot Keys Enabled")
  ...
  self~createPushButton(IDC_PB_RESET, 60, 135, 45, 15, , "Reset", onReset)
  ...

::method onReset

  ...
  editControl = self~newEdit(IDC_EDIT)

  if editControl~hasKeyPressConnection then
    self~newCheckBox(IDC_CHECK_FKEYSENABlED)~check
  else
    self~newCheckBox(IDC_CHECK_FKEYSENABlED)~uncheck
  ...
```

9.16. redrawRect

```
Form 1:

>>--redrawRect(--rectangle--+---------+--)----------------------><
                            +-,-erase-+

Form 2:

>>--redrawRect(--pt1--,--pt2--+---------+--)--------------------><
                              +-,-erase-+

Form 3:

>>--redrawRect(--x-,--y-,--x1-,--y1--+---------+--)-------------><
                                     +-,-erase-+

Generic form:

>>--redrawRect(--rectCoordinates--+---------+--)----------------><
                                  +-,-erase-+
```

The *redrawRect* method redraws the specified *bounding* rectangle within the client area of this dialog control. Optionally the background of the rectangle can be erased first.

Arguments:

The arguments are:

rectCoordinates [required]

The coordinates of the rectangle to be redrawn. The coordinates specify the upper left and lower right corners of the rectangle. The corners can be specified using either a *Rect* object, two *Point* objects, or the individual x and y coordinates of each corner.

The coordinates are specified as *client area* coordinates, in pixels.

erase [optional]

A boolean (**.true** or **.false**) that specifies whether the background of the rectangle should be redrawn (**.true**) or not **.false**.) The default is **.false**

Return value:

0 on success, 1 on error.

Remarks:

For the *erase* argument, the background is not really erased, but rather the background is specified to be redrawn also. By default the operating system does not redraw the background.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

9.17. setColor

```
>>--setColor(--+------+--+-------+--+---------+--)--------------><
               +-,-bk-+  +-,-fg--+  +-,-isClr-+
```

Sets the background color or foreground color, or both, for this dialog control.

Arguments:

The arguments are:

bk [optional]

Specifies the background color for this dialog control. This may either be a palette *color number*, or a *COLORREF* number. To use a COLORREF, the *isClr* argument must be true.

fg [optional]

Specifies the foreground color for this dialog control. This may either be a palette *color number*, or a *COLORREF* number. To use a COLORREF, the *isClr* argument must be true. The foreground color is the color that text is written in.

isClr [optional]

Specifies if the *bg* and *fg* arguments are to be interpreted as palette indexes or COLORREF numbers. The default if omitted is false.

Return value:

Returns 0 on success or 1 on error.

Remarks:

The *bk* and *fg* arguments must be specified in the same way. They must both be either palette indexes or both be COLORREF numbers. If the *bk* argument is omitted, the ooDialog framework ensures that the operating system paints the background using the same color as the dialog background.

Earlier versions of ooDialog had a restriction on the number of different dialog controls that could have their control changed from their default color. In ooDialog version 4.2.0 that restriction was lifted and there is no limit to the number of controls having their color set.

Example:

This example changes the text color of a static control in the dialog to a bright red using a COLORREF number:

```
::method changeColor
  expose staticText

  staticText~setColor( , .Image~colorRef(161, 6, 17), .true)
```

9.18. setParent

```
>>--setParent(--parent--)------------------------><
```

Sets a new parent for this dialog control.

Arguments:

The single argument is:

parent

The dialog control Rexx object that will be set as the parent window of this dialog control

Return value:

Returns true on success, false on error.

Remarks:

In Windows, the parent / child window relationship controls a number things, some having to do with how the windows are drawn. A child window is always drawn over the top of the parent, and no part of it can be drawn outside of the parent window. Historically in ooDialog, all dialog controls have be direct children of the dialog window. There are some advanced techniques that can make use of changing the parent of a dialog control to another control in the same dialog. This makes the dialog control the grandchild of the dialog.

Doing this causes the dialog control to be drawn over the top the new parent dialog control. Since the operating system handles the drawing automatically, the child dialog control appears as if it is part of the parent dialog control. The effect is much more professional looking than it would be if some other technique with ooDialog was used. The application of this method is of limited use however, and probably should only be used by Rexx programmers that have some knowledge of how Windows works.

Several considerations need to be taken into account when using this method. The programmer needs to understand that dialog controls send their event notifications to their parent window. This can not be changed. By setting another dialog control as parent of this dialog control, the programmer changes where this dialog control's notifications are sent. The connect event methods will no longer work with the control, because the notifications are no longer sent to the dialog. However, most all other methods of the dialog control should continue to work unchanged.

In order for event notifications from the grandchild dialog control to work, specialized code needs to be added to the ooDialog framework. See for instance the *isGrandchild* method which sets up some handling for an **`Edit`** control whose parent window is no longer the dialog. Future enhancements of ooDialog may add other specialized handling for grandchildren dialog controls. especially if ooDialog users request it and can show a generalized use for their request.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example sets the parent of an edit control to be a list-view in the dialog. When the user clicks on a subitem in the list-view, the edit control is position over the subitem and made visible. The user can type in the edit control to change the text of the subitem:

```
::method initDialog
  expose list edit

  list = self~newListView(IDC_LISTVIEW)

  edit = self~newEdit(IDC_EDIT)
  edit~setParent(list)
  edit~isGrandChild
  ...

::method onClick unguarded
  expose list edit editVisible lastIdx lastCol
  use arg id, itemIndex, columnIndex, keyState

  if lastIdx == itemIndex & lastCol == columnIndex then do
    if columnIndex > 0 then do
        r = list~getSubitemRect(itemIndex, columnIndex, 'LABEL')

        r~right  -= r~left
        r~bottom -= r~top
        flags = "SHOWWINDOW NOZORDERCHANGE"
```

```
      edit~setWindowPos(list~hwnd, r, flags)

      editVisible = .true
      edit~assignFocus
    end
  end

  lastIdx = itemIndex; lastCol = columnIndex

  return 0
```

9.19. setSysColor

```
>>--setSysColor(--+------+--+-------+--)---------><
                  +-,-bk-+  +-,-fg--+
```

Sets the background color or foreground color, or both, for this dialog control using the system *colors*.

Arguments:

The arguments are:

bk [optional]

The system color ID for the background color of the dialog control. This can be either the whole number ID or the keyword ID. IDs can be looked up in the System Color Elements *table*

fg [optional]

The system color for the foreground color of the dialog control. The foreground color is the color that text is written in. This can be either the whole number ID or the keyword ID. IDs can be looked up in the System Color Elements *table*.

Return value:

Returns 0 on success or 1 on error.

Remarks:

If the *bk* is omitted, the ooDialog framework ensures that the operating system paints the background of the control the same color as the background of the dialog. The *bk* and *fg* arguments must be specified in the same way. They must both be either palette indexes or both be COLORREF numbers.

Earlier versions of ooDialog had a restriction on the number of different dialog controls that could have their control changed from their default color. In ooDialog version 4.2.0 that restriction was lifted and there is no limit to the number of controls having their color set.

Example:

This example changes the background color of a static text lable in the dialog to the system background color for a menu:

```
::method changeColor
  expose label

  label~setSysColor('MENU')
```

9.20. setWindowTheme

```
>>--setWindowTheme(--themeName--)----------------><
```

Changes the visual theme for this dialog control window, causes the window to use a different set of visual style information than its class normally uses.

Arguments:

The single argument is:

themeName [required]

The application name to use in place of the calling application's name. See the remarks section.

Return value:

Returns true on success, false on error.

Remarks:

If this method is invoked on a *TreeView* object with the *themeName* of *explorer*, the theme for the tree-view is changed. The tree-view will use small triangles in place of the + and - buttons to expand and close the tree-view items. When invoked on a list-view with the *explorer* theme name, a change in the appearance of the list-view can be noticed. However, what other theme names are valid, or what other controls may have a change in appearance, is unknown to the ooDialog developers.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example updates a tree-view control to use the small triangle images for its expand buttons:

```
::method initDialog
  expose tv treeData

  tv = self~newTreeView(IDC_TREE)
  success = tv~setWindowTheme('explorer')
  if \ success then do
    say 'Error setting explorer theme.' .DlgUtil~errMsg(.systemErrorCode)
  end

  ...
```

9.21. tabStop

```
>>--tabStop(--+-----------+--)-------------------><
              +-wantStyle-+
```

Add or remove the *tabstop* style for this control.

Arguments:

The only argument is:

wantStyle [optional]

A boolean (.true or .false) to indicate whether the control should have or not have the tabstop style. True (the default) indicates the control should have the tabstop style and false indicates the control should not have the style.

Return value:

Negative values indicate the function failed, non-negative values indicate success.

-4 or less

The value is the negated Operating System Error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

-3

The second argument to the method is not a boolean.

-2

There is an (internal) problem with the dialog or the dialog handle.

-1

There is an (internal) problem with the resource ID of the control.

0 or greater

The window style of the dialog control prior to adding or removing the tabstop style.

Remarks:

When a control has the tabstop style, the user can set the focus to the control by using the tab key. When a control does not have this style, the tab key will skip over the control. Adding or removing this style during the execution of a dialog allows the programmer to alter how the user navigates through the dialog controls.

The negative return value from this method is a hold over from a period prior to the introduction of the *.systemErrorCode*. This method now sets the **`.systemErrorCode`**, making it easier to use the negative return as a failure indication and use the **`.systemErrorCode`** value to look up the actual error.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

9.22. textSize

```
>>--textSize(--text--,--size--)------------------><
```

Computes the width and height in pixels of the specified string of text when displayed by this control.

Arguments:

The arguments are:

text [required]

The text whose size is needed.

size [required] [in / out]

A Size *Size*) object. The computed size is returned in this argument.

Return value:

Returns true on success, otherwise false. It is unlikely this method will fail.

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example ...

```

```

9.23. useUnicode

```
>>--useUnicode(--use--)--------------------------><
```

Sets the Unicode character format flag that tells the dialog control to use, or not use, Unicode.

Arguments:

The single argument is:

use [required]

If this argument is **`.true`** the format flag is set to tell the control to use Unicode. If it is **`.false`** the format flag is set to not use Unicode.

Return value:

Returns the previous format flag for the control.

Remarks:

Currently, the ooDialog framework is compiled in ANSI only mode. Therefore this method will have no effect. It is possible that future version of ooDialog may be compiled in a manner that will allow Unicode to be used.

Details

Raises syntax errors when incorrect arguments are detected.

9.24. useVersion

```
>>--useVersion(--versionNumber--)----------------><
```

This method is used to inform the control that the programmer is expecting a behavior associated with a particular version common control library.

Arguments:

The single argument is:

versionNumber [required]

The common control version whose behavior this control is requested to use.

Return value:

Returns the version specified by the previous invocation of the *useVersion* method. If *versionNumber* is set to a value greater than the current version, returns -1.

Remarks:

In some cases, a control may behave in a different manner, depending on the version of the common control library. This mostly happens when bugs were fixed in later versions. The *useVersion* method allows the programmer to inform the control which behavior is expected. Which version behavior the control has been requested to use can be determined through the *usingVersion* method.

Note: If common control version 6 is in use, regardless of what value is set using *versionNumber*, *useVersion* returns version 6. Testing seems to indicate that if common control version 6 is in use, the *useVersion* method has no effect. The method is provided because the Microsoft documentation in many places suggests to use it with Custom Draw.

9.25. usingUnicode

```
>>--usingUnicode---------------------------------><
```

Determines if the dialog control is using Unicode characters.

Arguments:

There are no arguments for this method.

Return value:

The return is true if the dialog control is using Unicode characters and false if the control is not using Unicode characters.

Remarks:

Currently, the ooDialog framework is compiled in ANSI only mode. Therefore this method will always return false. It is possible that future version of ooDialog may be compiled in a manner that will allow Unicode to be used.

Details

Raises syntax errors when incorrect arguments are detected.

9.26. usingVersion

```
>>--usingVersion---------------------------------><
```

Returns the common control version number the control is using, if the *useVersion* method has been invoked by the programmer to set a specific version.

Arguments:

This method had no arguments.

Return value:

Returns the version set by the most recent invocation of the *useVersion* method for the control. Returns 0 if the method has never been invoked for the control.

Remarks:

The Microsoft documentation says, if no specific version has been set using the *useVersion* method, the control will return 0. However testing seems to indicate that documentation is incorrect or out of date. The documentation, in the remarks section, for the *useVersion* expands on this a little.

Button Controls

In Windows the button control has a number of different types or kinds. Most button types have a number of different styles. The style of a button effects its behavior and appearance. The appearance of a button is usually maintained by the operating system in combination with the programmer.

In general people refer to "pushing," "clicking," or "checking" a button. The mouse is used to click a button and the enter key to push a button. Checking a button can be done with the mouse or the keyboard. Buttons have a state, the most common of which are checked, unchecked, pushed, and focused.

The five kinds of buttons are:

- Push Buttons
- Check Boxes
- Radio Buttons
- Group Boxes
- Owner Drawn Buttons

ooDialog provides these classes to allow the programmer to interface with the *underlying* button controls:

Table 10.1. ooDialog Button Control Classes

Button Type	ooDialog Class
Push Button	*Button* class
Check Box Button	*CheckBox* class
Radio Button	*RadioButton* class
Group Box	*GroupBox* class
Owner Drawn Button	*AnimatedButton*, *Button*

10.1. Animated Buttons

The AnimatedButton class provides the methods to implement an animated button within a dialog. The attributes and methods are only described briefly in this document. An example program, **`oowalker.rex`**, is provided with the ooDialog sample programs.

ParentDlg
: Attribute holding the handle of the parent dialog

Stopped
: Animation ends when set to 1 (see Stop method)

init
: Initialize the animation parameters:

```
but = .AnimatedButton~new(buttonid,from,to, ,
                          movex,movey,sizex,sizey,delay, ,
                          startx,starty,parentdialog)
```

The values are stored in a stem variable:

sprite.buttonid
: ID of animation button

sprite.from
: Array of in-memory bitmap handles, or a bitmap resource ID in a DLL, or the name of an array in the .local directory containing handles to bitmaps loaded with *loadBitmap*. The array has to start with 1 and continue in increments by 1.

sprite.to
: 0 if sprite.from is an array, or the name of an array stored in **`.local`**, or a bitmap resource ID in a DLL

sprite.movex
: Size of one move horizontally (pixels)

sprite.movey
: Size of one move vertically

sprite.sizex
: Horizontal size of all bitmaps (pixels)

sprite.sizey
: Vertical size of all bitmaps

sprite.delay
: Time delay between moves (ms)

Startx and starty are the initial bitmap position, and parentdialog is stored in the ParentDlg attribute.

Two more values are initialized in the stem variable:

sprite.smooth
: Set to 1 for smooth edge change (can be changed to 0 for a bouncy edge change)

sprite.step
: Set to 1 as the step size between sprite.from and sprite.to for bitmaps in a DLL

SetSprite
: Set all the sprite. animation values using a stem:

```
mysprite.from = .array~of(bmp1,bmp2,...)
mysprite.to = 0
mysprite.movex = ...
...
self~setSprite(mysprite.)
```

GetSprite
: Retrieve the animation values into a stem:

```
self~getSprite(mysprite.)
```

SetFromTo
: Set bitmap information (sprite.from and sprite.to):

```
self~setFromTo(bmpfrom,bmpto)
```

SetMove

Set size of one move (sprite.movex and sprite.movey):

```
self~setMove(movex,movey)
```

SetDelay

Set delay between moves in milliseconds (sprite.delay):

```
self~setDelay(delay)
```

SetSmooth

Set smooth (1) or bouncy (0) edges (sprite.smooth):

```
self~setSmooth(smooth)  /* 1 or 0 */
```

setStep

Set the step size (sprite.step) between sprite.from and sprite.to for bitmaps in a DLL, for example, if bitmap resources are numbered 202, 204, 206, etc:

```
self~setFromTo(202,210)
self~setStep(2)
```

Run

Run the animation by going through all the bitmaps repetitively until dialog is stopped; invokes MoveSeq:

```
self~run
```

MoveSeq

Animate one sequence through all the bitmaps in the given move steps; invokes MovePos:

```
self~moveSeq
```

MovePos

Move the bitmaps by the arguments:

```
self~movePos(movex,movey)
```

MoveTo

Move the bitmaps in the predefined steps to the given position; invokes MoveSeq:

```
self~moveTo(posx,posy)
```

setPos

Set the new starting position of the bitmaps:

```
self~setPos(newx,newy)
```

getPos

Retrieve the current position into a stem:

```
self~getPos(pos.)
```

```
say "pos=" pos.x pos.y
```

ParentStopped
: Check the parent dialog window to see if it is finished. Returns 1 if the parent is finished.

Stop
: Stop animation by setting the stopped attribute to 1

HitRight
: Invoked by run when the bitmap hits the right edge (returns 1 and bitmap starts at left again; you can return 0 and set the new position yourself)

HitLeft
: Invoked when the bitmap hits the left edge (default action is to start at right again)

HitBottom
: Invoked when the bitmap hits the bottom edge (default action is to start at top again)

HitTop
: Invoked when the bitmap hits the top edge (default action is to start at bottom again)

To use an animated button a dialog has to:

- Define a button in a resource file (owner-drawn)
- Load the bitmaps of the animation into memory using an array
- Initialize the animated button with the animation parameters
- Invoke the run method of the animated button
- Stop the animation and remove the bitmaps from memory

The dialog may also dynamically change the parameters (for example, the size of a move, or the speed) and over-ride actions, such as hitting an edge.

See the **`oowalker.rex`** and **`oowalk2.rex`** examples in **`OODIALOG\SAMPLES`**.

For further information see *installAnimatedButton*.

10.2. Check Boxes

Check boxes are similar in many ways to radio buttons. A check box consists of a square box and a programmer defined label, icon, or bitmap that indicates to the user a choice. Check boxes are typically grouped together as a set of independent options. Checking or unchecking one option usually has no effect on the other check box options.

There are four styles of check boxes: standard, automatic, three-state, and automatic three-state, The system manages the check state of automatic and automatic three-state check boxes. All check boxes have at least two states, either checked or cleared (unchecked). In addition three-state check boxes have a third state called indeterminate, which the system draws as a grayed box inside the check box.

Repeatedly clicking standard and automatic check boxes toggles them back and forth from cleared to checked, and back again. Doing the same with a three-state check box toggles it from cleared, to checked, to indeterminate, and back again.

The **CheckBox** class provides methods to query and modify check box controls.

The **CheckBox** class is a subclass of the *Button* class and therefore has all the instance methods of that class. In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with check box controls:

Instantiation:

Use the *newCheckBox* method to retrieve an object of the check box class.

Dynamic Definition:

To dynamically define a check box in a *UserDialog* class, use one of the create button control methods described in *Section 8.6.3, “Create Button Controls”*. That section also describes methods for creating push buttons, radio buttons, etc... All the methods to create check boxes begin with: *createCheck*.

Event Notification

A check box, is a button, and the programmer uses the same method to receive notifications of events as with the button control, the *connectButtonEvent* method.

10.2.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with **CheckBox** objects, including the pertinent methods from other classes.

Table 10.2. CheckBox Instance Methods

Method	Description
Useful	**External Methods**
newCheckBox	Returns a **CheckBox** object for the control with the specified ID.
createCheckBox	Creates a check box in the dialog template of a *UserDialog*.
connectButtonEvent	Connects button event notifications to a method in the Rexx dialog object.
Instance Methods	**Instance Methods**
isIndeterminate	Determines if the check box is in indeterminate state, or not.
getCheckState	Returns a keyword indicating the check state of the check box, checked, unchecked, or indeterminate.
setIndeterminate	Puts the check box in the indeterminate state.

10.2.2. newCheckBox (dialog object method)

Check box objects can not be instantiated by the programmer from Rexx code. Rather a check box object is obtained by using the *newCheckBox* method of the *dialog* object. The syntax is:

```
>>--newCheckBox(--id--)--------------------------><
```

10.2.3. createCheckBox (UserDialog method)

A check box object can be created in the dialog template for a *UserDialog* dialog through the *createCheckBox* method. The basic syntax is:

```
>>--createCheckBox(-id-,--x-,--y--+------+--+------+--+---------+--+---------+->
                                  +-,-cx-+  +-,-cy-+  +-,-style-+  +-,-label-+

>----+----------------+--+-------------+--)---------------------------------><
```

```
      +-,-attributeName-+  +-,-connectOpt-+
```

10.2.4. getCheckState

```
>>--getCheckState--------------------------------><
```

This method overrides the RadioButton *getCheckState* method to return a keyword indicating the check state of the check box, checked, unchecked, or indeterminate.

Arguments:

There are no arguments to this method.

Return value:

The return value is a keyword indicating the checked state of the check box and will be exactly one of the following:

UNCHECKED	The check box is not checked.
CHECKED	The check box is checked.
INDETERMINATE	The check box is in the indeterminate state.
UNKNOWN	The object is not a check box.

Example:

This example is a variation of the example for the isIndeterminate() method.

```
state = self~newCheckBox(IDC_CHK_BACKGROUND)~getCheckState

select
  when state == "CHECKED" then return self~blackBackground
  when state == "UNCHECKED" then return self~whiteBackground
  when state == "INDETERMINATE" then return self~grayBackground
  otherwise nop  -- Drop through and do error handling
end
-- End select
```

10.2.5. isIndeterminate

```
>>--isIndeterminate------------------------------><
```

Determines if the check box is in indeterminate state, or not.

Arguments:

There are no arguments.

Return value:

Returns true if the check box is in the indeterminate state, otherwise false.

Example:

This example is from a program that sets the background color to black, white, or gray.

```
chkBox = self~newCheckBox(IDC_CHK_BACKGROUND)
```

```
    if chkBox~isIndeterminate then return self~grayBackground
```

10.2.6. setIndeterminate

```
>>--setIndeterminate------------------------------><
```

Puts the check box in the indeterminate state. The system will redraw the check box to indicate the indeterminate state.

Arguments:

This method takes no arguments.

Return value:

This method always returns 0.

10.3. Group Boxes

The Windows Group Box control is actually a button control, not a static control. Its purpose is to group together a set of related controls. It consists of a label and a rectangle. The related controls are placed within the rectangle. Group boxes have no state, they can not be selected, and an application can not send messages to the control. In addition, a group box does not send notifications to its parent so there are no events to connect to a group box.

The GroupBox class provides methods to work with group box controls.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with group box controls:

Instantiation:

Use the *newGroupBox* method to retrieve an object of the group box class.

Dynamic Definition:

To dynamically define a group box in the dialog template of a *UserDialog* class, use the *createGroupbox* method.

Event Notification

Group boxes do not generate notifications.

10.3.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with **`GroupBox`** objects, including the pertinent methods from other classes.

Table 10.3. GroupBox Instance Methods

Method	Description
Useful	**External Methods**
newGroupBox	Returns a **`GroupBox`** object for the control with the specified ID.
createGroupBox	Creates a group box in the dialog template of a *UserDialog*.
Instance Methods	**Instance Methods**
style=	Assigns a new text alignment to the group box.

10.3.2. newGroupBox (dialog object method)

Group box objects can not be instantiated by the programmer from Rexx code. Rather a group box object is obtained by using the *newGroupBox* method of the *dialog* object. The syntax is:

```
>>--newGroupBox(--id--)--------------------------><
```

10.3.3. createGroupBox (UserDialog method)

A group box object can be create in the dialog template for a *UserDialog* dialog through the *createGroupbox* method. The basic syntax is:

```
>>--createGroupbox(-+------+-,-x-,-y-,-cx-,-cy--+---------+-+--------+-)-------><
                    +--id--+                    +-,-style-+ +-,-text-+
```

10.3.4. style=

```
>>--style = styleKeyword-------------------------><
```

Assigns a new text alignment to the group box. By default the text of a group box is in the upper left corner. The alignment can also be set to the right or to the center, although this is not commonly done.

Arguments:

The only argument is:

styleKeyword

A keyword that is one of the following:

LEFT

The group box label is aligned to the upper left.

RIGHT

The label is aligned to the upper right.

CENTER

The label is aligned to the upper center.

Example:

Changing the style is straightforward:

```
gb = self~newGroupBox(IDC_GB_AREACODES)
if gb == .Nil then return
gb~style="RIGHT"
```

10.4. Radio Buttons

Radio buttons consist of a round button and a programmer defined label, icon, or bitmap that indicates to the user a choice. Radio buttons are typically grouped together as a set of mutually exclusive options. Radio buttons can be either standard or automatic. The system manages the check state of automatic radio buttons. The state of a radio button can be either checked or cleared (unchecked.)

provides methods to query and modify radio button controls.

The **RadioButton** class is a subclass of the *Button* class and therefore has all the instance methods of that class. In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with radio button controls:

Instantiation:

Use the *newRadioButton* method of the *dialog* object to retrieve a new **RadioButton** object.

Dynamic Definition:

To dynamically define a radio button in a *UserDialog* class, use the *createRadioButton* method. There are also a number of methods that create groups of radio buttons. These methods are described described in *Section 8.6.3, "Create Button Controls"*. That section also describes methods for creating push buttons, check boxes, etc..

Event Notification

To connect the *event* notifications sent by the underlying radio button control to a method in the Rexx dialog object use the *connectButtonEvent* method.

10.4.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with radio button objects, including the pertinent methods from other classes.

Table 10.4. RadioButton Class and Instance Methods

Method	Description
Useful	**External Methods**
newRadioButton	Returns a **RadioButton** object for the control with the specified ID.
createRadioButton	Creates a radio button in the dialog template of a *UserDialog*.
connectButtonEvent	Connects button event notifications to a method in the Rexx dialog object.
Class Methods	**Class Methods**
checkInGroup	Checks the specified radio button in a group and unchecks all other radio buttons in the group.
Instance Methods	**Instance Methods**
check	Puts the button in the checked state.
checked	Determines if the button is checked, or not.
getCheckState	Returns a keyword indicating the check state of the radio button.
uncheck	Puts the button in the unchecked state.

10.4.2. newRadioButton (dialog object method)

Radio button objects can not be instantiated by the programmer from Rexx code. Rather a radio button object is obtained by using the *newRadioButton* method of the *dialog* object. The syntax is:

```
>>--newRadioButton(--id--)----------------------><
```

10.4.3. createRadioButton (UserDialog method)

A radio button object can be created in the dialog template for a *UserDialog* dialog through the *createRadioButton* method. The basic syntax is:

```
>>--createRadioButton(-id-,-x-,-y-+------+--+------+--+---------+-------------->
                                  +-,-cx-+  +-,-cy-+  +-,-style-+

>--+--------+--+--------+--)--------------------------------------------------><
   +-,-text-+  +-,-attr-+
```

10.4.4. checkInGroup (Class)

```
>>--checkInGroup(--dlg--,--idFirst--,--idLast--+-----------+--)---------------><
                                               +-,-idCheck--+
```

Adds a check mark to (checks) the specified radio button in a group and removes the check mark from (clears) all other radio buttons in the group.

Arguments:

The arguments are:

dlg [required]

The dialog that contains the radio button to be checked.

idFirst [required]

The resource ID of the first radio button in the group. May be numeric or symbolic.

idLast [required]

The resource ID of the last radio button in the group. May be numeric or symbolic.

idCheck [optional]

The resource ID of the radio button that is to be checked. May be numeric or symbolic. If this argument is omitted, is 0, or -1 than all radio buttons in the group are unchecked.

Return value:

This method return 0 on success and a *system error code* if the operating system reports one.

Remarks:

The use of the *idCheck* argument allows the programmer to uncheck all radio buttons.

Raises syntax errors when incorrect arguments are detected.

Details:

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect arguments are detected.

Example:

This method is simple to use:

```
ret = .RadioButton~checkInGroup(self, IDC_RB_WHITE, IDC_RB_BLACK, IDC_GREEN)
return 0
```

10.4.5. check

```
>>--check----------------------------------------><
```

The check method puts the button in the checked state. The system will redraw the button with the check mark appropriate for the button.

Arguments:

This method takes no arguments.

Return value:

This method always returns 0.

10.4.6. checked

```
>>--checked--------------------------------------><
```

Determines if the button is checked, or not.

Arguments:

There are no arguments.

Return value:

Returns true if the button is checked, otherwise false.

Example:

This example determines if the user is choosing to print the current page, all pages, or a selection:

```
if self~newRadioButton(IDC_RB_CURRENT)~checked then do
  --Print the current page ...
end
else if self~newRadioButton(IDC_RB_ALL)~checked then do
  -- Print all pages ...
end
else do
  -- Must be print the selection ...
end
```

10.4.7. getCheckState

```
>>--getCheckState--------------------------------><
```

Returns a keyword indicating the check state of the radio button, checked or unchecked.

Arguments:

There are no arguments to this method.

Return value:

The return value is a keyword indicating the checked state of the radio button and will be exactly one of the following:

UNCHECKED	The radio button is not checked.
CHECKED	The radio button is checked.
UNKNOWN	The object is not a radio button.

Example:

This method is straightforward to use:

```
rb = self~newRadioButton(IDC_RB_ITALIC)
if rb~getCheckState == "CHECKED" then self~printInItalics()
...
```

10.4.8. uncheck

```
>>--uncheck--------------------------------------><
```

The uncheck method puts the button in the unchecked state. The system will redraw the button without the check mark.

Arguments:

This method takes no arguments.

Return value:

This method always returns 0.

10.5. Push Buttons

Push buttons are rectangular controls containing a label (a text string,) and an image defined by the programmer. The label or image indicates what the button does when it is pushed or clicked. A push button is either standard or default.

Standard push buttons usually start an operation. The button will receive the keyboard focus when the user clicks it. On the other hand, the default push button will indicate the default choice, which would normally be the most common choice. The default button does not have to have the input focus to be selected. It is selected by the user pressing the ENTER key when the dialog box has the focus, no matter which control currently has the input focus.

The **`Button`** class provides methods to query and modify all types of button controls, including push buttons. It is the superclass of both the *RadioButton* and *CheckBox* classes. The class also has methods that allow the Rexx programmer to partially implement owner drawn buttons using bitmaps.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with push buttons:

Instantiation:

Use the *newPushButton* method of the dialog *dialog*) object to retrieve a new **`Button`** object.

Dynamic Definition:

To dynamically define a push button control in a *UserDialog* class, use the *createPushButton* method. There are also a number of methods that create groups of push buttons. These methods are described described in *Section 8.6.3, “Create Button Controls”*. That section also describes methods for creating radio buttons, check boxes, etc..

Event Notification

To connect the *event* notifications sent by the underlying push button to a method in the Rexx dialog object use the *connectButtonEvent* method.

10.5.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with Button objects, including the pertinent methods from other classes:

Table 10.5. Button Methods

Method	Description
Useful	**External Methods**
newPushButton	Returns a **`Button`** object for the control with the specified ID.
createPushButton	Creates a button control in the dialog template of a *UserDialog*.
connectButtonEvent	Connects button event notifications to a method in the Rexx dialog object.
Class Methods	**Class Methods**
Instance Methods	**Instance Methods**
click	Simulates a user clicking on the associated button control.
getIdealSize	Queries the button control for the size that best fits its text or image.
getImage	Retrieves a button's image if one is set.
getImageList	Retrieves information describing the image list for the button.
getTextMargin	Retrieves the margins used to draw text within the button.
push	Simulates the user pushing the associated button control.
setImage	Sets, or removes, the image associated with a button.
setImageList	Sets, or removes the image list for the button.
setTextMargin	Sets the margins for drawing text in the button.
state	Retrieves the current state of the button.
state=*state=*)	Sets the state for the button.
style=	Changes the style of the button.

10.5.2. newPushButton (dialog object method)

Push button objects can not be instantiated by the programmer from Rexx code. Rather a push button object is obtained by using the *newPushButton* method of the *dialog*object. The syntax is:

```
>>--newPushButton(--id--)------------------------><
```

10.5.3. createPushButton (UserDialog method)

A push button object can be created in the dialog template for a *UserDialog* dialog through the *createPushButton* method. The basic syntax is:

```
>>--createPushButton(-id-,-x-,-y-,-cx-,-cy-,-+---------+-+-------+-+-------+-)-><
                                             +-,-style-+ +-,-txt-+ +-,-mth-+
```

10.5.4. connectButtonEvent (dialog object method)

To connect event notifications from a button object use the *connectButtonEvent* method of the *dialog* object. The basic syntax is:

```
>>--connectButtonEvent(--id--,--event--+---------------+--)-------------------><
```

```
                                          +--,-methodName--+
```

10.5.5. click

```
>>--click----------------------------------------><
```

The *click* method programmatically clicks the associated button control. It simulates the user clicking the button and produces exactly the same behavior as if the user had clicked on the button with the mouse.

Arguments:

This method takes no arguments.

Return value:

This method always returns 0.

Example:

This example closes the dialog as if the user had clicked the cancel button. (Provided of course that the dialog has a cancel button.)

```
self~newPushButton(IDCANCEL)~click
```

10.5.6. getIdealSize

```
>>--getIdealSize(--+-------------+--)------------><
                   +--wantWidth--+
```

This method queries the button control for its ideal size. The ideal size is the size that best fits its text and image. (If the button has an image.)

Arguments:

The single optional argument is:

wantWidth [optional]

Specifies the desired width of the button in pixels. If this argument is used, the operating system will calculate the idea height for a button width the width specified. This functionality is not available on Windows XP

Return value:

The possible return values are:

A *Size* object

The ideal size for the button in pixels.

The **.nil** object

Some unanticipated error. This is very unlikely to happen.

Details:

This method requires Common Control *Library* version 6.0 or later.If necessary use the *comCtl32Version* method to determine the current version of the library.

Raises syntax errors when incorrect usage is detected.

Example:

```
size = self~newPushButton(IDC_PB_REVIEW)~getIdealSize
say 'The ideal height for the button is' size~height
say 'The ideal width for the button is' size~width

/* Output might be for example:
 *   The ideal height for the button is 24
 *   The ideal width for the button is 45
 */
```

10.5.7. getImage

```
>>--getImage-------------------------------------><
```

Retrieves a button's image if one is set, or **`.nil`** if the button does not have an image associated with it.

Arguments:

This method does not take any arguments.

Return value:

This method returns the *Image* object for the button, or .nil if the button does not have an image.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

In this example a bitmap has been set for the button on a popup dialog. When the dialog closes, bitmap image is retrieved from the button and released.

```
::method cancel
  button = self~newPushButton(IDC_PB_PUSHME)
  image = button~getImage
  if image <> .nil then image~release
  return self~cancel:super
```

10.5.8. getImageList

```
>>--getImageList---------------------------------><
```

This method retrieves information describing the image list for the associated button, if there is one.

Arguments:

There are no arguments to this method.

Return value:

The return value is:

A **`.Directory`** object.

The image list information is returned in a Directory object. See *setImageList* for clarification of the information returned. The directory object has the following entries:

imageList

The *ImageList* object containing the images for the button.

rect
: A *Rect* object that specifies the margin around the image.

alignment
: The numeric flag that specifies the alignment of the image list.

alignmentKeyword
: The string keyword that specifies the alignment of the image list. This will be exactly one of: Left, Right, Top, Bottom, Center, or Unknown. It is very unlikely that the keyword will ever be Unknown.

The `.nil` object
: The Nil object is returned if the button does not have an image list, or for some other unanticipated error.

Details:

This method requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version* method to determine the current version of the library.

Raises syntax errors when incorrect arguments are detected.

Example:

This example gets the image list from the View push button and displays the information returned.

```
  pbView = self~newPushButton(IDC_PB_VIEW)
  d = pbView~getImageList

  say 'Got image list:' d
  if d <> .nil then do
    say '  image:   ' d~imageList
    say '  rect:    ' d~rect
    say '  align:   ' d~alignmentKeyword

    say 'Image margins:' d~rect~left',' d~rect~top',' d~rect~right',' d~rect~bottom
  end
  ...

/* Output might be for example:

  Got image list: a Directory
    image:   an ImageList
    rect:    a Rect
    align:   Left
  Image margins: 1, 1, 1, 1

 */
```

10.5.9. getTextMargin

```
>>--getTextMargin--------------------------------><
```

This method retrieves the margins used to draw text within the button control.

Arguments:

This method does not take any arguments.

Return value:

The possible return values are:

A *Rect* object

The margin for drawing text in the button.

The **`.nil`** object.

An unanticipated error. This is not likely to happen.

Details:

This method requires CommonControl *Library* version 6.0 or later. If necessary use the *comCtl32Version* method to determine the current version of the library.

Raises syntax errors when incorrect usage is detected.

Example:

This example gets the text margins for the Redraw button and displays the information:

```
margins = self~newPushButton(IDC_PB_REDRAW)~getTextMagin
say 'The text margins for the "Redraw" button:'
say '  Left:  ' margins~left
say '  Top:   ' margins~top
say '  Right: ' margins~right
say '  Bottom:' margins~bottom

/* Output might be for example:
 *   The text margins for the "Redraw" button:
 *     Left:   1
 *     Top:    1
 *     Right:  1
 *     Bottom: 1
 */
```

10.5.10. push

```
>>--push-----------------------------------------><
```

The *push* method simulates the user pushing the associated button control. It allows a Rexx programmer to produce the exact same behavior from within an ooDialog program as the behavior produced by the user pushing the button in the dialog.

Arguments:

There are no arguments to this method.

Return value:

This method always returns 0.

Example:

This example comes from an imaginary program that has a list of window handles. The dialog has a Refresh button that the user can push to update the window list. The code comes from a section of the program that is checking if a specific window handle is valid. If the handle is not valid, the code simulates the user pushing the Refresh button to update the window list.

```
::method validateHandle private
    use strict arg hwnd
    if \ self~isWindowHandle(hwnd) then do
        self~newPushButton(IDC_PB_REFRESH)~push
        return .false
    end
    ...
return .true
```

10.5.11. setImage

```
>>--setImage(-newImage-)-------------------------><
```

Sets, or removes, the image associated with a button. The image can bet a bitmap, icon, or cursor. Note that the button style has to match the type of image. For cursors and icons, the button must be an ICON button. For bitmap images, the button must be a BITMAP button.

To completely remove an image associated with the button use **`.nil`** for the argument.

Arguments:

The single argument is:

newImage [required]

The new*Image* object to be associated with the button or **`.nil`**. The use of **`.nil`** for the argument removes any existing image.

Return value:

This method returns the old image, if there was one, otherwise **`.nil`**.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

This example gets an image from the resource file for a *ResDialog* and associates it with the Push Me button.

```
::method initDialog
  module = .ResourceImage~new("imageButton.dll", self)
  image = module~getImage(101)
  self~newPushButton(IDC_PB_PUSHME)~setImage(image)
  self~connectButtonEvent(IDC_PB_PUSHME, "CLICKED", onPushMe)
```

10.5.12. setImageList

```
>>--setImageList(--imageList--,--+--------+--,--+-------+--)-------------------><
                                 +-margin-+     +-align-+
```

Sets, or removes, the image list for the button. To remove an existing image list, the programmer passes in .nil for the first argument.

Arguments:

The arguments are:

imageList [required]

A *ImageList* object containing the images for the button. If this argument is the **`.Nil`** object, then any existing image list is removed.

margin [optional]
: A *Rect* object containing the margins around the image. The default is a 0 margin.

alignment [optional]
: Specifies the alignment of the image on the button. The *alignment* argument can be specified either by the numeric value of the flag that the button control recognizes, or by a string keyword.

The following keywords can be used to specify the alignment, case is not significant:

LEFT	RIGHT
TOP	BOTTOM
CENTER	

To specify the alignment using a numeric value, the programmer can use the *toID* method of the *Image* class to get the correct numeric value for one of the following symbols, or use the numeric value itself.

BUTTON_IMAGELIST_ALIGN_LEFT	BUTTON_IMAGELIST_ALIGN_RIGHT
BUTTON_IMAGELIST_ALIGN_TOP	BUTTON_IMAGELIST_ALIGN_BOTTOM
BUTTON_IMAGELIST_ALIGN_CENTER	

The default is CENTER, of BUTTON_IMAGELIST_ALIGN_CENTER if using the numeric value.

Return value:

The possible return values are:

oldImageList
: If there is an existing image list, it is returned in the same format specified in *getImageList*.

.nil
: The **`.Nil`** object is returned if there was no existing image list, and also if an error is encountered. In general, the **`.systemErrorCode`** variable should be set to non-zero if an error happened.

Remarks:

Using an image list with a button is a new feature that requires Windows XP or later. It provides an easier, more convenient way to do owner-drawn buttons. This method should be the preferred way for the ooDialog programmer to do bitmap buttons, where the primary purpose is to provide an image on the button. Using an image list will automatically give the button the same look and feel as other buttons on the system.

A *ImageList* object is used to supply the images for the button. If the image list only contains 1 image at the first index, then that image is used for all button states. If more than 1 image is supplied, then the image at each index is used for the button state with this mapping:

index 0 = Normal
index 1 = Hot
index 2 = Press
index 3 = Disabled
index 4 = Defaulted
index 5 = StylusHot

StylusHot is only used on tablet computers. If more than 1 image is supplied, but some indexes do not have an image, then the system does not draw an image for that state. The programmer retains ownership of the image list. What this means in essence is the programmer decide when, or if, the image list should be released. See the discussion in the *ImageList* class about releasing image lists if needed. If you release the image list while the dialog is still active, the images on the button disappear.

A sample program: **`imageButton.rex`** in the samples directory is provided that shows how to use this method. It is in: **`samples\oodialog\examples`**.

Details:

This method requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version* method to determine the current version of the library.

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect arguments are detected.

Example:

This example creates an image list from a set of bitmap files. Each image represents one of the possible buttons states. In the program, as the button changes states, the text on the button is also changed. So, the program calculates the ideal button size using the longest label, then resets the button size to the ideal size.

Since the image list API is only available with XP or later, the program checks that the API is available before proceeding. For further information on some of the methods used in the example see *getIdealSize* and *comCtl32Version*.

```
  ...
  if .DlgUtil~comCtl32Version  < 6 then return -2

  files = .array~new()
  files[1] = "resources\Normal.bmp"       -- Normal
  files[2] = "resources\Hot.bmp"          -- Hot (hover)
  files[3] = "resources\Pushed.bmp"       -- Pushed
  files[4] = "resources\Disabled.bmp"     -- Disabled
  files[5] = "resources\Default.bmp"      -- Default button
  files[6] = "resources\Hot.bmp"          -- Stylus hot, tablet PC only

  -- Set the flags to create a 24 bit color, masked image list.
  flags = "COLOR24 MASK"
  imageList = .ImageList~create(.Size~new(61, 46), flags, 10, 10)

  images = .Image~fromFiles(files)
  cRef = .Image~colorRef(255, 255, 255)
  imageList~addImages(images, cRef)

  align = LEFT
  margin = .Rect~new(1)

  ret = pbView~setImageList(imageList, margin, align)
  if .systemErrorCode <> 0 then do
    -- put some error handling here
    return -1
  end

  -- Temporarily set the title to the longest text, to calculate
  -- the ideal size.
  pbView~setTitle("DEATH if you touch me")
  bestSize = pbView~getIdealSize

  -- Not reset the button label and set the size to the ideal size.
  pbView~setTitle("View Pictures")
  pbView~setRect(0, 0, bestSize~width, bestSize~height, "NOMOVE")

return 0
```

10.5.13. setTextMargin

```
>>--setTextMargin(--margins--)-------------------><
```

This method sets the margins for drawing text in the associated button control.

Arguments:

The only argument is:

margins

A *Rect* object describing the new margins.

Return value:

The possible return values are:

.true

Success.

.false

Failed.

Details:

This method requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version* method to determine the current version of the library.

Raises syntax errors when incorrect arguments are detected.

Example:

This example sets a new text margin of 5 for the button control.

```
margins = .Rect~new(5, 5, 5, 5)
self~newPushButton(IDC_PB_SHOW_DETAILS)~setTextMargin(margins)
```

10.5.14. state

```
>>--state----------------------------------------><
```

The *state* method retrieves the current state of the associated button control.

Arguments:

This method takes no arguments.

Return value:

A text string that can contain one or more of the following keywords, separated by blanks:

CHECKED

The radio button or the check box is checked. A radio button is checked when it contains a black dot, a check box is checked when it contains a check mark.

UNCHECKED

The radio button or the check box is not checked. A radio button is not checked when it does not contain a black dot and a check box is not checked when it does not contain a check mark.

INDETERMINATE
: A 3-state check box button is neither checked nor unchecked. Only 3-state check box buttons can be in this state. When in the indeterminate state, the check box button is drawn in a visually distinctive manner that is different from checked or unchecked. Exactly how the indeterminate state is drawn is dependent on the operating system version.

PUSHED
: When a button is in the pushed state it is drawn as a sunken button, otherwise it is drawn as a raised button. The user causes a button to be in the pushed state by clicking the button with the left mouse button. As long as the mouse button is held down, the button will be drawn as pushed. When the user releases the mouse button, the button will resume its not pushed state. When this keyword is not in the text string, the button is not in the pushed state.

FOCUS
: The button has the keyboard focus. When this keyword is missing from the text string, the button does not have the focus.

Example:

```
button = MyDialog~newPushButton("IDOK")
if button == .Nil then return
say button~State
```

The result could be "UNCHECKED FOCUS".

10.5.15. state=

```
>>--state = newState-------------------------------><
```

The *state*= method sets the state for the associated button control.

Arguments:

The only argument is:

newState [required]
: A text string that contains one or more of the following keywords, separated by a blank:

CHECKED	UNCHECKED
PUSHED	NOTPUSHED
INDETERMINATE	FOCUS

CHECKED
: The radio button or the check box is to be set to the "checked" state.

UNCHECKED
: The radio button or the check box is to be set to the "unchecked" state.

PUSHED
: The button is to be set to the "pushed" state.

NOTPUSHED
: The "pushed" state is to be removed from the button.

INDETERMINATE
: The 3-state check box button is to be set to the "indeterminate" state. This state can only be applied to 3-state check box buttons.

FOCUS
: The button is to be set to the "focused" state.

Example:

```
button = MyDialog~newPushButton("IDOK")
if button == .Nil then return
button~state="FOCUS PUSHED"
```

Remarks:

Some points to remember when using this method.

1. The checked, unchecked or indeterminate states have no meaning for a push button. Setting a push button to one of these states therefore has no effect.

2. A radio button or check box can be set to only one of the checked, unchecked or indeterminate states. If more than one of these states is specified in the text string the button is set to the state that is first in the string.

3. Only 3-state check box buttons can be set to the indeterminate state. If this keyword is used for a button that is not a 3-state button then the button's state is not changed.

4. To change a button state to not focused programmatically use one of the ooDialog methods that move the focus: *tabToNext*, *tabToPrevious*, *setFocus*, *focusControl*, etc..

10.5.16. style=

```
>>--style = newStyle-----------------------------><
```

This method assigns a new style to the *underlying* button control.

Arguments:

The only argument is:

newStyle [required]
: A text string containing one or more of the following keywords separated by blanks. Common sense should be used when constructing the new *style* string. If mutually exclusive key words are used, the outcome is undefined.

DEFPUSHBUTTON	TEXT	NOTPUSHLIKE
LEFTTEXT	MULTILINE	AUTO3STATE
BOTTOM	CHECKBOX	RIGHT
PUSHBOX	ICON	NOTMULTILINE
RIGHTBUTTON	NOTIFY	GROUPBOX
VCENTER	AUTOCHECKBOX	HCENTER
RADIO	BITMAP	NOTNOTIFY
NOTLEFTTEXT	FLAT	OWNERDRAW
PUSHLIKE	3STATE	TOP
AUTORADIO	LEFT	NOTFLAT

DEFPUSHBUTTON
: In a dialog, the default push button is the button that is pushed when the Enter key is pressed. Changing the default push button will change the behavior of the dialog when the user presses the enter key.

PUSHBOX
: A push button that does not display the button face or border, only the text appears. This button style is not applicable when Windows themes are in use.

RADIO
: A radio button the state of which has to be maintained by the programmer.

AUTO
: A radio button where the operating system maintains the check state for all radio buttons in the same group. When one radio button in the group is checked by the user, the operating system unchecks all other buttons in the group.

CHECKBOX
: A check box where the state has to be maintained by the programmer. The state can only be checked or unchecked.

AUTOCHECKBOX
: A check box where the operating system maintains the check state for the programmer.

3STATE
: A check box button that has a grayed state as well as checked or unchecked. The greyed state shows that the check state of the button is not determined. The programmer needs to manage the state of the button when it is clicked.

AUTO3STATE
: A check box button that is the same as a three-state check box except that the operating system maintains the check state for the programmer.

GROUPBOX
: This key word has no effect.

OWNERDRAW
: This key word has no effect.

LEFTTEXT
: Places the text on the left of the button for a radio button or check box. This style and RIGHTBUTTON are equivalent.

RIGHTBUTTON
: Places the button on the right of the text for a radio button or a check box. This style is the same as LEFTTEXT.

NOTLEFTTEXT
: Removes the LEFTTEXT style.

TEXT
: The button displays text.

ICON
: The button displays an icon image.

BITMAP
: The button displays a bitmap image.

LEFT
: The button has its text left-justified in the button rectangle.

RIGHT
: The button has its text right-justified in the button rectangle.

HCENTER
: The button has its text centered horizontally in the button rectangle.

TOP
: The text is placed at the top of the button rectangle.

BOTTOM
: The text is placed at the bottom of the button rectangle.

VCENTER
: The text is vertically centered in the button rectangle.

PUSHLIKE
: Causes a radio button or check box button to behave like a push button.

MULTILINE
: Causes the text for a button to wrap to multiple lines if the text is too long for the width of the button.

NOTIFY
: Enables a button to send the kill focus and set focus *event* notifications. See the *connectButtonEvent* method of the *EventNotification* class. If the button does not have this style, kill and set focus events will not be received, even if the programmer does use the *connectButtonEvent* method to connect those events.

FLAT
: Gives the button a flat appearance. This style is not applicable when Windows themes are in effect.

NOTPUSHLIKE
: Removes the push like style.

NOTMULTILINE
: Removes the multi-line style.

NOTNOTIFY
: Removes the notify style.

NOTFLAT
: Removes the flat style.

Remarks:

In the past, the documentation for this method was incomplete or misleading. There are really two aspects to a button, its type (or kind) and its style. Many of the button styles only have meaning within the same type of button. These styles can not be applied to a button of a different type after the button has been created.

For instance, the AUTOCHECKBOX style can only be applied to a check box type of button. Trying to give this style to a radio type button has no effect. Because of this, certain of the style key words have no effect. They are listed here only because they were listed in previous versions of the documentation. This may have lead someone to use the key word in their code. An example is the GROUPBOX key word. The group box type of button only has one style. This style can not be applied to any other type of button, so setting the style of any button to GROUPBOX has no effect, and has never had any effect.

However, within each type of button, the styles that apply to that type of button can be changed. For instance, a check box type of button that has the 3STATE style, can be changed to have the AUTOCHECKBOX style or to have the AUTO3STATE style.

Example 1:

The following example makes the OK button the default button:

```
button = MyDialog~newPushButton("IDOK")
if button == .Nil then return
button~style="DEFPUSHBUTTON"
```

Example 2:

This example changes the text of a check box button depending on the state another check box. When the button text is long, the multi-line style is added and the placement of the text in relation to the button rectangle is changed. When the text is short, the button style is set back to the default style.

```
  ::method idCheckOne
    use arg wParam, lParam

    id = .DlgUtil~loWord(wParam)
    if self~newCheckBox(id)~checked then do
      chkButton = self~newCheckBox(IDC_CHECK_TWO)
      chkButton~style = "MULTILINE TOP HCENTER"
      chkButton~setTitle("Use IPv6 not IPv4 during this connection")
    end
    else do
      chkButton = self~newCheckBox(IDC_CHECK_TWO)
      chkButton~style = "NOTMULTILINE VCENTER LEFT"
      chkButton~setTitle("Connect")
    end
```

Combo Box Controls

There are 3 types of combo boxes.

- Simple combo boxes
- Drop-down combo boxes
- Drop-down list combo boxes

A combo box consists of a list and a selection field. The list presents options that can be selected and the selection field displays the current selection.

In simple and drop-down combo boxes the selection field is an edit control and can be used to enter text not available in the list, or to edit existing text. In a drop-down list combo box, the list is a list box control and the use can only select the items in the list box. In drop-down and drop-down list combo boxes the list only appears when the user opens it. In a simple combo box the list is always visible.

The **`ComboBox`** class provides methods to query and modify combo box controls.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with combo box controls:

Instantiation:
: Use the *newComboBox* method of the *dialog* object to retrieve a combo box object.

Dynamic Definition:
: To dynamically create a combo box in the dialog template of a *UserDialog* use the *createComboBox* method.

Event Notification
: To connect the *event* notifications sent by the underlying combo box control to a method in the Rexx dialog object use the *connectComboBoxEvent* method.

11.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with ComboBox objects, including the pertinent methods from other classes:

Table 11.1. ComboBox Methods and Attributes

Method	Documentation
Useful External Methods	
newComboBox	Obtains a ComboBox object that represents a combo box control in a dialog.
createComboBox	Creates a combo box control in an *UserDialog*.
connectComboBoxEvent	Connects ComboBox event notifications to a Rexx dialog method.
Instance Methods	
add	Adds a new item to the list of this combo box.
addDirectory	Adds file names in a given directory to the items of this combo box.
closeDropDown	Closes the drop-down list of this combo box.
delete	Removes an item from the list of this combo box.

Method	Documentation
deleteAll	Removes all list items from this combo box.
find	Searches this combo box for a list item containing the specified text or prefix.
getCue	Retrieves the cue banner text for this combo box, or the empty string if there is no cue set.
getCombBoxInfo	Returns a **`Directory`** object containing information about this combo box.
getEditControl	Returns a Rexx **`Edit`** object that represents the child edit control used by this combo box.
getFirstVisible	Determines the one-based index of the top item in the drop-down list.
getHorizontalExtent	Gets the value of the horizontal extent for the combo box.
getItemHeight	Determines the height of the list items or the height of the selection field in this combo box.
getMinVisible	Retrieves the minimum number of visible items that must be displayed in the drop-down list of this combo box.
getText	Gets the text of the list item at the specified position in this combo box.
insert	Inserts a new item into the list of the combo box after the specified item.
isDropDown	Determines if this combo box is a drop-down combo box.
isDropDownList	Determines if this combo box is a drop-down list combo box.
isDropDownOpen	Determines if the list of this combo box is dropped down.
isGrandchild	Notifies the ooDialog framework that this combo box control is a grandchild of the dialog and configures the underlying combo box control to send four event notifications to the dialog, rather than its direct parent.
isSimple	Determines if this combo box is a simple combo box.
items	Retrieves the number of items in the combo box.
modify	Changes the text of the list item at the specified position in the combo box.
openDropDown	Drops down the list of the combo box.
removeFullColor	Reverts the color for this combo box to its default, if the *setFullColor* method has previously been invoked on this combo box.
select	Selects the list entry that matches the specified text.
selected	Retrieves the text of the currently selected item in the combo box list.
selectedIndex	Retrieves the index of the currently selected item in the combo box list.
selectIndex	Selects the list item at the specified index.
setCue	Sets the cue banner text for this combo box.
setEditSelection	Selects the specified text range in the edit control of the combo box.
setFullColor	Sets the complete color of this combo box, this includes both the selection field and the list parts of the comb box.
setHorizontalExtent	Sets the width, in pixels, that the combo box's list box can be scrolled horizontally.
setItemHeight	Sets the height of the list items or the selection field in this combo box.

Method	Documentation
setMinVisible	Sets the minimum number of visible items in the drop-down list of a combo box.

11.2. newComboBox (dialog object method)

`ComboBox` objects can not be instantiated by the programmer from Rexx code using a *new* method. Rather, a combo box object is obtained by using the *newComboBox* method of the *dialog* object. The syntax is:

```
>>-newComboBox(--id--)---------------------------><
```

11.3. createComboBox (UserDialog method)

A combo box control can be created in the dialog template of a *UserDialog* through the *createComboBox* method. The basic syntax is:

```
>>-createComboBox(-id-,--x-,--y-,--cx-,--cy-+----------+-+-------------+--)---><
                                            +-,--style-+ +-,--attrName-+
```

11.4. connectComboBoxEvent (dialog object method)

To connect event notifications from a combo box control use the *connectComboBoxEvent* method of the *dialog* object. The basic syntax is:

```
>>-connectComboBoxEvent(--id--,--event--+---------------+--)------------------><
                                        +--,-methodName--+
```

11.5. add

```
>>--add(--listEntry--)---------------------------><
```

The *add* method adds a new item to the combo box list.

Arguments:

The only argument is:

listEntry [required]

The item to add to the list.

Return value:

On success, the one-based index of the new item. On error, a value less than 1.

Remarks:

If the combo box does not have the SORT style, the new item is added to the end of the list. If it does have the SORT style, the item is inserted into the list in its correct alphabetical position.

Note that the Microsoft documentation says that adding an item to a comb box with the SORT style causes the list of items to be re-sorted. This does not appear to be correct. Rather, it looks like the operating system simply picks the correct insertion point for the new item and leaves the rest of the list alone.

Example:

The *find example* uses the *add* method.

11.6. addDirectory

```
>>--addDirectory(--filePattern--+-------------------+--)--------><
                                +-,-fileAttributes--+
```

Adds the names of all files matching *filePattern* that have the optionally specified file attributes to the combo box.

Arguments:

The arguments are:

filePattern [required]

A file name pattern of files to search for. For example, **`C:\Program Files\*.exe`**, or **`C:\temp\*.txt`**. The *fileAttributes* argument can be used to further restrict the types of files searched for. All matching file names are added to the combo box.

fileAttributes [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. The keywords specify the file attributes the files must possess in order to be added to the combo box:

READWRITE	SYSTEM	EXCLUSIVE
READONLY	DIRECTORY	DRIVES
HIDDEN	ARCHIVE	

READWRITE

Includes read write files with no additional attributes. This is the default if the *fileAttributes* argument is omitted.

READONLY

Includes read-only files.

HIDDEN

Includes hidden files.

SYSTEM

Includes system files.

DIRECTORY

Includes subdirectories. The subdirectory names are enclosed in square brackets ([ ]) when they are added to the combo box.

ARCHIVE

Includes archived files.

EXCLUSIVE

Includes only the files with the specified attributes. By default, read write files are listed even if READWRITE is not specified.

DRIVES
: Mapped drives are also added to the list. The drives names are added in the form [-D-], where D is the drive letter.

Return value:

The one-based index of the file name added last to the combo box, or less than 1 if an error occurred.

Remarks:

The *filePattern* argument can be a file name, relative path, or absolute path. It can contain the normal file name wildcard characters. Absolute paths can begin with a drive letter or use Universal Naming Convention (UNC). I.e. **`D:\`** or **`\\server\share`**.

If a file or directory name matches the file pattern and has the attributes specified by *fileAttributes* it is added to the list displayed by the combo box.

Note that the operating system searches for and adds the files to the combo box. What files are found and how they are displayed is up to the operating system. ooDialog has no control over that.

Example:

The following example puts the names of all the read/write files with extension **`.rex`** in the **`C:\work.ooRexx`** directory into the combo box:

```
cb = self~newComboBox(IDC_CB_FILES)
self~addDirectory("C:\work.ooRexx\*.rex", "READWRITE")
```

11.7. closeDropDown

```
>>--closeDropDown--------------------------------><
```

Closes the drop-down list box of a drop-down combo box or a drop-down list combo box. This method has no effect on simple combo boxes.

Arguments:

This method has no arguments.

Return value:

This method always returns **`.true`**

11.8. delete

```
>>--delete(--+---------+--)----------------------><
             +--index--+
```

Removes an item from the list in the combo box.

Arguments:

The only argument is:

index [optional]
: The one-based index of the item to delete from the list. If this argument is omitted, the currently selected item is deleted.

Return value:

The number of items remaining in the list. -1 is returned if *index* is not valid.

Remarks:

If *index* is omitted and there is no selected item, no item is deleted and -1 is returned.

11.9. deleteAll

```
>>--deleteAll------------------------------------><
```

Removes all items from the list in a combo box.

Arguments:

This method has no arguments.

Return value:

This method always returns **`.true`**

11.10. find

```
>>--find(--textOrPrefix--+-------------+--+---------+--)-------><
                         +-,-startIndex-+  +-,-exact-+
```

Finds the index of the combo box item that matches the specified text.

Arguments:

The arguments are:

textOrPrefix [required]

The text of the item to search for.

startIndex [optional]

The one-based index of the item preceding the item to start the search at. If the search reaches the end of the items without a match, the search continues with the first item until all items have been examined. When this argument is omitted, or 0, the search starts with the first item.

exact [optional]

Whether to do an exact match or not. If this argument is not omitted, and is either **`.true`** or the keyword EXACT, then an exact search is performed. Otherwise, the search is not exact. See the Remarks section for the meaning of exact and not exact.

Return value:

The one-based index of the item when found, otherwise 0 if the item is not found.

Remarks:

The search for *textOrPrefix* is **always** case insensitive. If the *exact* argument does not specify an exact match, then the search is for an item beginning with the text specified. Otherwise, the characters of the item, including blanks, must match the specified text exactly.

Example:

This, somewhat contrived, example demonstrates how the arguments to *find* control the search:

```
::method initDialog
  expose comboBox

  comboBoxItems = .array~of("San Diego", "New York", "Los Angeles",  -
                            "Detroit", "Tampa Bar", "Billings",      -
                            "San Francisco", "new orleans", "San Antonio")


  comboBox = self~newComboBox(IDC_CBO_ONE)
  do i over comboBoxItems
    comboBox~add(i)
  end

::method search private
  expose comboBox

  self~report(comboBox~find("San"),                    comboBox)
  self~report(comboBox~find("San", 1),                 comboBox)
  self~report(comboBox~find("San", 7),                 comboBox)

  self~report(comboBox~find("New", , .true),           comboBox)
  self~report(comboBox~find("New", , .false),          comboBox)
  self~report(comboBox~find("New Orleans", , "exact"), comboBox)


::method report private
  use strict arg index, comboBox

  if index = 0 then say "Not found"
  else say comboBox~getText(index)

/* Output would be:

San Diego
San Francisco
San Antonio
Not found
New York
new orleans

*/
```

11.11. getCue

```
>>--getCue---------------------------------------><
```

Retrieves the cue banner text for this combo box, or the empty string if there is no cue set.

Arguments:

There are no arguments for this method.

Return value:

Returns the cue for this combo box, or the empty string if no cue is set.

Remarks:

The cue banner is text that is displayed in the edit control, or selection field, of a combo box when there is no item currently selected.

Details

Requires Windows Vista or later.

Raises syntax errors if incorrect usage is detected.

Example:

This example sets a simple cue for a combo box and then reads it back:

```
  comboBox = self~newComboBox(IDC_COMBOBOX)

  comboBox~setCue("Select a city")

  say 'Cue set:' comboBox~getCue

/* Output would be:

   Cue set: Select a city

*/
```

11.12. getCombBoxInfo

```
>>--getCombBoxInfo-------------------------------><
```

Returns a **`Directory`** object containing information about this combo box.

Arguments:

There are no arguments with this method.

Return value:

On error the **`.nil`** object is returned. On success a **`Directory`** object is returned whose indexes contain information concerning this combo box. The directory object will have the following indexes set to the values indicated:

TEXTOBJ:

For simple and drop-down combo boxes this index contains the Rexx *Edit* object that represents the child edit control of the combo box. Drop-down list combo boxes do not have a child edit control, and this index will contain the Rexx combo box object itself. (The operating system returns the combo box for this value.)

LISTBOXOBJ:

A Rexx *ListBox* object that represents the child list box control of the combo box.

TEXTRECT:

A *Rect* object that contains the coordinates of the edit control. For drop-down list controls, the Microsoft documentation does not specify what the returned coordinates are, but experimentation shows that it seems to be the area that the edit control occupies in simple of drop-down combo boxes.

BUTTONRECT:

A *Rect* object that contains the coordinates of the button that contains the drop-down arrow.

BUTTONSTATE:

A keyword that indicates the button state. The keyword will be exactly one of the following words:

notpressed	absent	pressed

notpressed
: The button exists and it is not pressed.

absent
: There is no button. This would be the case for the simple combo box.

pressed
: The button exists and it is pressed.

Details

Sets the *.SystemErrorCode* variable.

Example:

This example shows a method that prints out the information for a simple combo box:

```
::method printInfo
    expose cbSimple

    d = cbSimple~getComboBoxInfo
    say 'Simple Combo Box'
    say '  Button rect: ' d~buttonRect
    say '  Text rect:   ' d~textRect
    say '  Text object: ' d~textObj
    say '  List object: ' d~listBoxObj
    say '  Button state:' d~buttonState
    say

/* Output might be for example: */

Simple Combo Box
  Button rect:  a Rect (1, 1, -1, -1)
  Text rect:    a Rect (3, 3, 101, 18)
  Text object:  an Edit
  List object:  a ListBox
  Button state: absent
```

11.13. getEditControl

```
>>--getEditControl-------------------------------><
```

Returns a Rexx **`Edit`** object that represents the child edit control used by this combo box.

Arguments:

There are no arguments for this method

Return value:

Returns an **`Edit`** object that represents the edit control used by this combo box. Returns the **`.nil`** object on error. Note that drop-down list combo boxes do not have an edit control.

Remarks:

The returned edit control is a grandchild of the dialog so the *connectEditEvent* method can not connect any events to the Rexx dialog. Dialog controls only send their notifications to their direct parent. However, the *isGrandchild* method can be used to set up special handling so that a few events can be intercepted and used to invoke a method in the Rexx dialog.

Because drop-down list combo boxes do not have a child edit control, this method does not work with drop-down list combo boxes.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example gets the child edit control of the combo box so that the application can manipulate the edit control. It also uses the *isGrandchild* so that the application can monitor some of the edit control's event notifications:

```
cb = self~newComboBox(IDC_CB)

edit = cb~getEditControl
edit~isGrandchild
```

11.14. getFirstVisible

```
>>--getFirstVisible------------------------------><
```

Determines the one-based index of the top item in the drop-down list.

Arguments:

There are no arguments for this method.

Return value:

The one-based index of the item at the top of the list on success. O is returned on error.

Remarks:

Initially the item with index 1 is at the top of the list. However, if the user has scrolled the list, the top index may no longer number 1.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example calculates the top y coordinate of the item under the mouse by subtracting the top index from the index of the item under the mouse and multiplying that by the individual height:

```
topIndex = comboBox~getFirstVisible
r = lbRect~copy
r~top = (currentItem - topIndex) * itemHeight
```

11.15. getHorizontalExtent

```
>>--getHorizontalExtent--------------------------><
```

Gets the width that the list box of the combo box can be scrolled horizontally.

Arguments:

There are no arguments for this method.

Return value:

The return value is the width in pixels of the horizontal extent.

Remarks:

This method is only applicable if the combo box has a horizontal scroll bar. A scroll bar can be added explicitly by using the WS_HSCROLL style with the combo box definition in the resource script for a *RcDialog* or a *ResDialog*, or by using the HSCROLL style in the *createComboBox* method of a *UserDialog*.

In addition, a horizontal scroll bar is added implicitly if the CBS_DISABLENOSCROLL style is used in the resource script. (Or the DISABLENOSCROLL style in the *createComboBox* method.)

When the *underlying* combo box is created, it has its horizontal extend set to 0. The *setHorizontalExtent* method can be used to change this value.

11.16. getItemHeight

```
>>--getItemHeight(--+--------------------+--)----><
                    +--getSelectionField-+
```

Determines the height of the list items or the height of the selection field in this combo box.

Arguments:

The single arguments is:

getSelectionField [optional]

If true get the selection field height, if false get the list items height. The default is false, get the item height.

Return value:

Returns the height, in pixels, of the specified component. On error returns -1.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example gets the height of the combo box items and increases it by 2 pixels:

```
itemHeight = comboBox~getItemHeight
itemHeight += 2
comboBox~setItemHeight(itemHeight)
```

11.17. getMinVisible

```
>>--getMinVisible--------------------------------><
```

Retrieves the minimum number of visible items that must be displayed in the drop-down list of this combo box.

Arguments:

There are no arguments for this method.

Return value:

Returns the minimum number of visible items.

Remarks:

When the number of items in the drop-down list is greater than the minimum, the combo box uses a scrollbar. Unless the combo box has the no integral height style, the drop-down will be sized so that it can accommodate the minimum number of items. By default, 30 is the minimum setting for the combo box.

Details

Requires Common Control *Library* version 6.0 or later.

Raises syntax errors when incorrect usage is detected.

Example:

This example queries the current minimum visible setting, and then changes it to 10:

```
  say 'Current minimum visible setting:' comboBox~getMinVisible
  comboBox~setMinVisible(10)
  say 'New minimum visible setting is: ' comboBox~getMinVisible

/*  Output would be:

Current minimum visible setting: 30
New minimum visible setting is:  10

*/
```

11.18. getText

```
>>--getText(--index--)---------------------------><
```

Gets the text of the item at the specified index in the combo box list.

Arguments:

The only argument is:

index [required]

The one-based index of a combo box item.

Return value:

The text of the combo box item at the specified index. If *index* is not valid or an error occurs, the empty string is returned.

Example:

The *find example* uses the *getText* method.

11.19. insert

```
>>--insert(--+-------+--,--item--)---------------><
```

```
              +-index-+
```

Inserts a new item into the list of the combo box.

Arguments:

The arguments are:

index [optional]

The one-based index of the position to add the new item. If this *index* is omitted, the item is added after the currently selected item. If there is no currently selected item, the new item is inserted as the last item. If *index* is 0 the item is inserted as the first item. If *index* is less than 0 then the new item is inserted as the last item

item [required]

The item to add to the list.

Remarks:

Unlike the *add* method, if the combo box has the SORT style, the *insert* method does not cause the combo box to sort the list items.

If the combo box has a horizontal scroll bar and the width of the new item is wider than the combo box, or wider than the current horizontal extent, the programmer should use the *setHorizontalExtent* method to ensure the scroll bar is shown and will scroll far enough to show the whole item.

Return value:

On success, the one-based index of the position at which the item was been added. A number less than 1 indicates an error.

11.20. isDropDown

```
>>--isDropDown-----------------------------------><
```

Determines if this combo box is a drop-down combo box.

Arguments:

There are no arguments for this method.

Return value:

Returns true if this combo box is a drop-down combo box, otherwise false.

Remarks:

Recall there are 3 types of combo boxes. A *simple* combo box, a *drop-down* combo box, and a *drop-down list* combo box. This method determines if this combo box is a *drop-down* combo box.

11.21. isDropDownList

```
>>--isDropDownList-------------------------------><
```

Determines if this combo box is a drop-down list combo box.

Arguments:

There are no arguments for this method.

Return value:

Returns true if this combo box is a drop-down list combo box, otherwise false.

Remarks:

Recall there are 3 types of combo boxes. A *simple* combo box, a *drop-down* combo box, and a *drop-down list* combo box. This method determines if this combo box is a *drop-down list* combo box.

11.22. isDropDownOpen

```
>>--isDropDownOpen------------------------------><
```

Determines whether the list box of the combo box is dropped down.

Arguments:

This method has no arguments.

Return value:

Returns **`.true`** if the list box is dropped down, otherwise **`.false`**.

11.23. isGrandchild

```
>>--isGrandchild(--+------------+--+------------+--)------------><
                   +-,-mthName--+  +-,-wantTab--+
```

Notifies the ooDialog framework that this combo box control is a grandchild of the dialog and configures the underlying combo box control to send four event notifications to the dialog, rather than its direct parent.

The four events that the grandchild notifies the Rexx dialog of are the RETURN, ESCAPE, TAB keydown events, and the KILLFOCUS event. See the Remarks section for more details.

Arguments:

The arguments are:

mthName [optional]

The name of the method in the Rexx dialog that is connected to the four event notifications sent by the grandchild. If this argument is omitted, the default method name is *onComboBoxGrandchildEvent*.

wantTab [optional]

By default, the TAB keydown notification is not sent to the dialog. If the *wantTab* argument is true then the *mthName* method is also invoked for the TAB keydown event. The default for this argument is false and the TAB keydown event does not cause the method to be invoked.

Return value:

Returns true on success, false on error.

Remarks:

Normally dialog controls are child windows of the dialog window. Historically, in older versions of ooDialog, this was always the case. However, since ooDialog 4.2.0, there have begun to be some cases where the programmer can gain access to dialog controls that are not the direct child of the dialog. For instance, the *setParent* method can be used to change the parent window of a dialog control to a parent window other than the dilaog window. When the other window is a dialog control, this makes the control window a grandchild of the dialog. Some dialog controls create their own child dialog controls and enhancements to ooDialog have begun to give the Rexx programmer access to these child dialog controls. These are also grandchildren of the dialog.

A dialog control sends its event notifications to its direct parent, never to its grandparent. The *isGrandchild* method of the **ComboBox** control is a special case method. Its purpose is to allow the ooDialog framework to set up some special handling so that four event notifications will be intercepted and used to invoke a method in the Rexx dialog. Three of the methods are keydown events, the TAB, RETURN, and ESCAPE keydown events. These events occur when the user types the return, (or enter,) key, the escape key, or the tab key. The fourth event is the KILLFOCUS event. This event occurs when the combo box control loses the focus.

Note that this method will probably only work properly for drop-down list combo boxes. The simple and drop-down combo boxes have a child edit control that receives the keyboard focus rather than the combo box. For simple and drop-down combo boxes, the programmer should be get the edit control using the *getEditControl* method and then use the **Edit** class *isGrandchild* method on the edit object rather than the *isGrandchild* method on the combo box object.

Details

Requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version*() method to determine the current version of the library. A syntax error is raised if this method is invoked on an operating system where it is not supported.

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example sets the parent of a combo box to be a list-view. It then invokes *isGrandchild* so that the *onComboBoxGrandchildEvent* method will be invoked in the Rexx dialog for the KILLFOCUS, RETURN keydown, and ESCAPE keydown events:

```
list = self~newListView(IDC_LISTVIEW)
cb   = self~newComboBox(IDC_COMBOBOX)

cb~setParent(list)
cb~isGrandChild
```

11.24. isSimple

```
>>--isSimple-------------------------------------><
```

Determines if this combo box is a simple combo box.

Arguments:

There are no arguments for this method.

Return value:

Returns true if this combo box is a drop-down list combo box, otherwise false.

Remarks:

Recall there are 3 types of combo boxes. A *simple* combo box, a *drop-down* combo box, and a *drop-down list* combo box. This method determines if this combo box is a *simple* combo box.

11.25. items

```
>>--items----------------------------------------><
```

Returns the number of items in the combo box list.

Arguments:

This method has no arguments.

Return value:

The number of items in the combo box list, or -1 on error. An error is very unlikely.

11.26. modify

```
>>--modify(--+--------+--,--newText--)-----------><
             +--index-+
```

Replaces the text for the item at the specified index with *newText*.

Arguments:

The arguments are:

index [optional]

The index of the list item whose text is to be replaced. If this argument is omitted, then the currently selected item has its text replaced. When this argument is omitted and there is no currently selected text then nothing is done and -1 is returned.

newText [required]

The new text for the item at the specified index.

Return value:

The one-based index of the item whose text has been replaced, or less than 1 to indicate an error.

Remarks:

Because this method uses the *insert* method to change the text at the specified index, if the combo box has the SORT style, the *modify* method will not cause the combo box to insert the new text in its alphabetical order in the list items.

If the combo box has a horizontal scroll bar and the width of the new text is wider than the combo box, or wider than the current horizontal extent, the programmer should use the *setHorizontalExtent* method to ensure the scroll bar is shown and will scroll far enough to show the whole item.

11.27. openDropDown

```
>>--openDropDown---------------------------------><
```

Shows the drop-down list box of a drop-down combo box or a drop-down list combo box. This method has no effect on simple combo boxes.

Arguments:

This method has no arguments.

Return value:

This method always returns `.true`

11.28. removeFullColor

```
>>--removeFullColor------------------------------><
```

Reverts the color for this combo box to its default, if the *setFullColor* method has previously been invoked on this combo box.

Arguments:

This method takes no arguments.

Return value:

Returns true on success, otherwise false.

Remarks:

If the *setFullColor* method has not previously been invoked for this combo box, then the method does nothing and false is returned.

Details

Raises syntax errors when incorrect usage is detected.

Requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version*() method to determine the current version of the library. A syntax error is raised if this method is invoked on an operating system where it is not supported.

Example:

This example restores the combo box to its default colors after it had been set to some custom colors:

```
::method initDialog
    ...
    mediumSpringGreen =  .Image~colorRef(0, 238, 118)
    fireBrickRed      =  .Image~colorRef(205, 38, 38)

    cbDropDown = self~getComboBox(IDC_CB_DROPDOWN)
    cbDropDown~setFullColor(mediumSpringGreen, fireBrickRed)
    ...

::method restoreComboBox private
    expose cpDropDown

    cbDropDown~removeFullColor
    return 0
```

11.29. select

```
>>--select(--itemText--)-------------------------><
```

Sets the selected item in the list of the combo box to the first item whose text matches *itemText*

Arguments:

The single argument is:

itemText [required]
: The text to use in searching for the matching item to select.

Return value:

The one-based index of the item selected, or zero if no match was found. 0 is also returned if an error occurs within the *underlying* combo box , but that is unlikely.

Remarks:

The search for *itemText* is always case insensitive. The search starts with the first item in the list and the first item whose beginning characters match *itemText* is selected. For example, if *itemText* is **`san`** then the item **`San Diego`** would be a match.

If no match is found, or if an error occurs, the selection remains unchanged.

Example:

This example fills a combo box with items and then selects an item.

Notice that the result(s) as specified in the code comments, are dependent on the combo box not having the SORT style. If the combo box does have the sort style, then in both cases *new orleans* would be selected.

```
::method initDialog
  expose comboBox

  comboBoxItems = .array~of("San Diego", "New York", "Los Angeles",  -
                            "Detroit", "Tampa Bar", "Billings",      -
                            "San Francisco", "new orleans", "San Antonio")


  comboBox = self~newComboBox(IDC_CBO_ONE)
  do i over comboBoxItems
    comboBox~add(i)
  end

  comboBox~select("new") -- Selects New York

  /* On the other hand this would select new orleans */
  -- comboBox~select("NEW O")
```

11.30. selected

```
>>--selected-------------------------------------><
```

Returns the text of the currently selected item in the list of the combo box.

Arguments:

This method has no arguments.

Return value:

The text of the selected item, or the empty string if no item is selected.

11.31. selectedIndex

```
>>--selectedIndex--------------------------------><
```

Determines the index of the currently selected item in the list of the combo box.

Arguments:

This method has no arguments.

Return value:

The one-based index of the currently selected list item, or 0 if there is no selected item.

11.32. selectIndex

```
>>--selectIndex(--+---------+--)-----------------><
                  +--index--+
```

Sets the currently selected item in the list of the comboBox to the index specified.

Arguments:

The only argument is:

index [optional]

The one-based index of the item to be selected.

If this argument is omitted, or is 0, the selection is cleared. I.e., the result will be that no item is selected.

Return value:

On success, the return is the one-based index of the selected item, which is 0 if the selection has been cleared. On error, -1 is returned.

Remarks:

When a new item is selected, if necessary, the new item in the list is scrolled into view. The text in the selection field will change to reflect the new selected item. Any previous selection in the list is removed.

If *index* is greater than the number of items in the list, the selection is cleared and -1 is returned. If *index* is negative, then -1 is returned and the selection remains unchanged.

11.33. setCue

```
>>--setCue(--banner--)---------------------------><
```

Sets the cue banner text for this combo box.

Arguments:

The single argument is:

banner [required]

The text for the banner. The text can not be longer than 255 characters in length.

Return value:

Returns 0 on success or 1 on error.

Remarks:

The cue banner is text that is displayed in the edit control, or selection field, of a combo box when there is no item currently selected.

Details

Requires Windows Vista or later.

Raises syntax errors when incorrect usage is detected.

Example:

This example sets a simple cue for a combo box and then reads it back:

```
  comboBox = self~newComboBox(IDC_COMBOBOX)

  comboBox~setCue("Select a city")

  say 'Cue set:' comboBox~getCue

/* Output would be:

   Cue set: Select a city

*/
```

11.34. setEditSelection

```
>>--setEditSelection(--+--------------+--+------------+--)-----><
                       +--startIndex--+  +-,-endIndex--+
```

Sets the selection in the edit control portion of a *simple* or drop-down combo box.

Arguments:

The arguments are:

startIndex [optional]

The one-based index of the starting character for the selection. If *startIndex* is 0, the selection, if any, is removed. The default value for *startIndex* is 0.

endIndex [optional]

The one-based index of the last character in the selection. If *endIndex* is 0 all text in the edit control is selected. The default value if *endIndex* is omitted is 0.

Return value:

This method returns 0 on success and 1 on error. In particular, if this method is used with a drop-down list combo box, 1 is returned. Other errors could occur within the *underlying* combo box, but they are unlikely.

Remarks:

If the value of *startIndex* is 0, the value of *endIndex* is ignored. This implies that if *startIndex* is omitted, the selection is always removed.

If *endIndex* is 0 and the value of *startIndex* is not omitted and is not 0, then all text in the edit control is selected and the actual value of *startIndex* is not relevant.

endIndex can be less than *startIndex*. If both arguments are positive then text is selected, but the selection is exclusive of the staring and ending indexes. For example this code:

```
cb = self~newComboBox(200)
ret = cb~setEditSelection(8, 3)
```

will select characters 4 through 7 inclusive. Provided there are at least 7 characters in the edit control of course.

The outcome of other variations of *endIndex* being less than *startIndex* are deterministic, but the results are left as an exercise for the reader.

11.35. setFullColor

```
>>--setFullColor(--+-----------+--+------------+--+---------------+--)---------><
                   +--bkColor--+  +-,-fgColor--+  +-,-isSysColor--+
```

Sets the complete color of this combo box, this includes both the selection field and the list parts of the comb box.

Arguments:

The arguments are:

bkColor [optional]

Specifies the background color for the combo box. If this arugment is omitted, the background color of the combo box is not changed. See the remarks for the details on how colors are specified. If both the *bkColor* and the *fgColor* arguments are omitted, this method returns false and does not do anything.

fgColor [optional]

Specifies the foreground color for the combo box. If this arugment is omitted, the foreground color of the combo box is not changed. See the remarks for the details on how colors are specified. If both the *bkColor* and the *fgColor* arguments are omitted, this method returns false and does not do anything.

isSysColor [optional]

The colors can be either regular colors or system colors, but both the foreground and the background colors must be the same type. The *isSysColor* argument is used to specify that the colors are system colors. By default, the *bkColor* and *fgColor* arguments are assumed to be regular colors specified as a *COLORREF*. If *isSysColor* is true then the colors are assumed to be system colors.

Return value:

Returns true on success and false on error.

Remarks:

Since a combo box is not a single control, but rather is composed of several different controls, the *setColor* and *setSysColor* methods do not work well with a combo box. Typically only the selection field is colored and the color for the list part of the combo box is left unchanged. The *setFullColor* method is used to set the color for all of the combo box.

Regular colors are specified as a *COLORREF* value, not as a palette *color number*. To ensure that the COLORREF value is correct, be sure to use one of the ooDialog methods that returns a COLORREF, such as the *colorRef* class method of the *Image* class.

System colors are specified to the operating by an unique whole number ID. To specify a system color to the *setFullColor* method, the programmer can either use the unique whole number ID known to the operating system or a keyword, case insensitive. The keywords, a brief description, and the matching whole number IDs can be looked up in the System Color Elements *table*.

The *bkColor* and the *fgColor* arguments must both be system color IDs or both be COLORREF values.

Details

Raises syntax errors when incorrect usage is detected.

Requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version*() method to determine the current version of the library. A syntax error is raised if this method is invoked on an operating system where it is not supported.

Example:

This example sets the color for a drop down combo box to light green on red:

```
mediumSpringGreen =  .Image~colorRef(0, 238, 118)
fireBrickRed      =  .Image~colorRef(205, 38, 38)

cbDropDown = self~getComboBox(IDC_CB_DROPDOWN)
cbDropDown~setFullColor(mediumSpringGreen, fireBrickRed)
```

11.36. setHorizontalExtent

```
>>--setHorizontalExtent(--pixels--)--------------><
```

Sets the width, in pixels, that the combo box's list box can be scrolled horizontally.

Arguments:

The single argument is:

pixels [required]

Specifies the scrollable width, in pixels, of the combo box's list box.

Return value:

This method always returns 0.

Remarks:

This method works whether or not the combo box has a horizontal scroll bar. However, if the combo box does not have a scroll bar, the user will see no difference. A scroll bar can be added explicitly by using the WS_HSCROLL style with the combo box definition in the resource script for a *RcDialog* or a *ResDialog*, or by using the HSCROLL style in the *createComboBox* method of a *UserDialog*.

In addition, a horizontal scroll bar is added implicitly if the CBS_DISABLENOSCROLL style is used in the resource script. (Or the DISABLENOSCROLL style in the *createComboBox* method.)

When the width of the list box in the combo box is equal to or greater than the horizontal extent value, the horizontal scroll bar is hidden. If there are items in the combo box that are wider than the width of the list box, the user will not be able to see all of the item. In this case, the horizontal extent needs to be set by the programmer to a higher value.

When the width of the list box in the combo box is smaller than the value of the horizontal extent, the horizontal scroll bar will horizontally scroll items in the list box.

Note that if the combo box has the DISABLENOSCROLL style, instead of hiding the horizontal scroll bar when the width of the list box is equal to or greater than the horizontal extent, the scroll bar is disabled.

Example:

This example calculates the width in pixels of the longest item added to the combo box. If this width is longer than the width of the combo box, then the horizontal extent is set to that width. Otherwise the horizontal extent is not changed. This will have no effect unless the combo box has a horizontal scroll bar, see the Remarks section.

```
::method initDialog
  expose comboBox comboBoxItems

  longest = .Size~new

  comboBox = self~newComboBox(IDC_CBO_ONE)
  do i over comboBoxItems
    comboBox~add(i)

    size = self~getTextSizePx(i)
    if size~width > longest~width then longest = size
  end

  -- Be sure we have a 1 pixel space at the end of the longest item.
  longest~width += 1

  if comboBox~pixelCX < longest~width then do
    comboBox~setHorizontalExtent(longest~width)
  end
```

11.37. setItemHeight

```
>>--setItemHeight(--height--+----------------------+--)---------><
                            +-,-setSelectionField--+
```

Sets the height of the list items or the selection field in this combo box.

Arguments:

The arguments are:

height [required]

The height, in pixels, to set the specified component. *height* must be a non-negative whole number or a syntax condition is raised.

setSelectionField [optional]

By default this method sets the height of the combo box items in the list. If *setSelectionField* is true, the height of the selection field is set.

Return value:

Returns 0 on success. Returns -1 if *height* is not valid. The Microsoft documentation does not specify exactly what invalid means. Some simple testing shows that the operating system will not accept very large numbers. Exactly where the cut off is could be determined by the programmer if needed.

Remarks:

The selection field height in a combo box is set independently of the height of the list items. The programmer must ensure that the height of the selection field is not smaller than the height of a particular list item.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example gets the height of the combo box items, increases it by 2 pixels, and sets that as the height of the items:

```
itemHeight = comboBox~getItemHeight
itemHeight += 2
comboBox~setItemHeight(itemHeight)
```

11.38. setMinVisible

```
>>--setMinVisible(--number--)--------------------><
```

Sets the minimum number of visible items in the drop-down list of a combo box.

Arguments:

The single argument is:

number [required]

The new setting for the minimum number of visible items in the drop-down list.

Return value:

Returns true on success, false on error.

Remarks:

When the number of items in the drop-down list is greater than the minimum, the combo box uses a scrollbar. Unless the combo box has the no integral height style, the drop-down will be sized so that it can accommodate the minimum number of items. By default, 30 is the minimum setting for the combo box.

Details

Requires Common Control *Library* version 6.0 or later.

Raises syntax errors when incorrect usage is detected.

Example:

This example queries the current minimum visible setting, and then changes it to 10:

```
  say 'Current minimum visible setting:' comboBox~getMinVisible
  comboBox~setMinVisible(10)
  say 'New minimum visible setting is: ' comboBox~getMinVisible

/*  Output would be:

Current minimum visible setting: 30
New minimum visible setting is:  10

*/
```

Date Time Picker Controls

A date and time picker (DTP) control provides a simple interface with which to exchange date and time information with a user. For example, with a DTP control you can ask the user to enter a date and then easily retrieve the selection.

Note

Windows does not support dates prior to 1601. Trying to set a date prior to 1601, programmatically, will raise a syntax condition.

The control is based on the Gregorian calendar, which was introduced in 1753. The date and time picker control will not calculate dates that are consistent with the Julian calendar.

The client area of a date and time picker (DTP) control displays date or time information, or both, and acts as the interface through which users modify the information. The date can be selected from a calendar, or by using an up-down control. The time can be changed by typing in fields that are defined by the control's Format *Strings*. The control can optionally display a check box. When it is checked, the value in the control is considered valid and can be retrieved. When not checked, the control's date and time is not valid.

The **`DateTimePicker`** class provides methods to get and set the date and time in the underlying DTP control, along with methods to manipulate the control. It is a concrete subclass of the dialog *control* object and therefore has all methods of the of the dialog control object.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with DTP controls:

Instantiation:
: Use the *newDateTimePicker* method of the *dialog* to retrieve a date and time picker object.

Dynamic Definition:
: To dynamically add a date and time picker to a *UserDialog* use the *createDateTimePicker* method.

Event Notification
: To receive notification of date and time picker events use the *connectDateTimePickerEvent* method.

12.1. Format Strings

Date and time picker controls rely on a *format* string to display the date and time. The format string defines how the date and time is displayed. There are 4 preset format strings that are assigned with 4 of the date and time styles. If the preset format strings are not adequate, the programmer can define his own format string and assign it to the date and time picker through the *setFormat* method.

The 4 preset format strings and their associated styles are in the following table:

Table 12.1. DTP Preset Formats

Style	Format
LONG	The format displays like: Wednesday, June 22, 2011.
SHORT	The format displays like: 6/22/11.

Style	Format
CENTURY	The format displays like: 6/22/2011.
TIME	The format displays like: 5:31:42 PM.

DTP format strings consist of a series of elements that represent a particular piece of information and define its display format. The elements are displayed in the order they appear in the format string. Custom formats give the programmer greater flexibility within a program. They allow the programmer to specify the order in which the control will display fields of information. The format string can include body text as well as callback fields for requesting information from the user.

Date and time format elements are replaced by the actual date and time. The following table shows the date and time elements and a description of what they produce:

Table 12.2. DTP Format Elements

Element	Description
d	The one- or two-digit day
dd	The two-digit day. Single-digit day values are preceded by a zero.
ddd	The three-character weekday abbreviation.
dddd	The full weekday name.
h	The one- or two-digit hour in 12-hour format.
hh	The two-digit hour in 12-hour format. Single-digit values are preceded by a zero.
H	The one- or two-digit hour in 24-hour format.
HH	The two-digit hour in 24-hour format. Single-digit values are preceded by a zero.
m	The one- or two-digit minute.
mm	The two-digit minute. Single-digit values are preceded by a zero.
M	The one- or two-digit month number.
MM	The two-digit month number. Single-digit values are preceded by a zero.
MMM	The three-character month abbreviation.
MMMM	The full month name.
t	The one-letter AM/PM abbreviation (that is, AM is displayed as *A*.)
tt	The two-letter AM/PM abbreviation (that is, AM is displayed as *AM*.)
yy	The last two digits of the year (that is, 2011 would be displayed as *11*.)
yyyy	The full year (that is, 2011 would be displayed as *2011*.)

Body text can be added to the format string by enclosing it in single quotes. Spaces and punctuation marks do not need to be quoted. **Note:** *Nonformat characters that are not delimited by single quotes will result in unpredictable display by the DTP control.* To include a single quote in the body text, use two consecutive single quotes.

Examples:

```
dtp = self~newDateTimePicker(IDC_DTP)

formatStr = "'Today is: 'hh':'m':'s dddd MMM dd', 'yyyy"

dtp~setFormat(formatStr)

/* The display would look like, depending on the actual date:
   Today is: 04:22:31 Wednesday Mar 02, 2011
*/

formatStr = "'Don''t forget' MMM dd',' yyyy"

dtp~setFormat(formatStr)

/* The display would look like, depending on the actual date:

   Don't forget Mar 02, 2011
*/

-- Quotes for the comma are not needed so this
-- is also valid and produces the same output:
formatStr = "'Don''t forget' MMM dd, yyyy"

dtp~setFormat(formatStr)

/* The display would look like, depending on the actual date:

   Don't forget Mar 02, 2011
*/
```

12.2. Callback Fields

The date and time picker control supports *call back fields* within the *format* strings. The call back fields allow the programmer to display custom information at the point of the call back field by allowing the control to query the application for information. For instance, the control can query the application for what text to display in the call back field or the maximum length the display text in a field will be.

The queries are done through *event* notification messages from the control. Specifically, the FORMAT and FORMATQUERY events connected using the *connectDateTimePickerEvent*. The programmer must respond to the event notification to ensure that the custom information is displayed properly. The response comes from the *FORMAT* and *FORMATQUERY* event handlers.

Call back fields are declared using one or more capital **X** characters (ASCII Code 88) in the format string. Unique callback fields are created by repeating the **X** character. Thus, the format string: **`"XX dddd MMM dd', 'yyy XXX"`** contains two call back fields. **XX** and **XXX**. When the FORMAT and FORMATQUERY notifications are sent, they include the format string element that defined the call back field so that the programmer can tell which call back field the notification is for.

12.3. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with DateTimePicker objects, including the pertinent methods from other classes:

Table 12.3. DateTimePicker Methods and Attributes

Method	Documentation
Useful External Methods	
newDateTimePicker	Obtains a DateTimePicker object that represents a date and time picker control in a dialog.
createDateTimePicker	Creates a DateTimePicker control in an *UserDialog*.
connectDateTimePickerEvent	Connects DateTimePicker event notifications to a Rexx dialog method.
Instance Methods	
closeMonthCal	Closes the drop down month calendar if it is dropped down

Method	Documentation
getDateTime	Retrieves the currently selected date and time for the DTP and returns it as a **`DateTime`** object.
getIdealSize	Gets the size needed to display the date time picker control without clipping.
getInfo	Returns a **`Directory`** object with information about the date time picker.
getMonthCal	Returns the *MonthCalendar* object for a date and time picker's (DTP) child month calendar control.
getMonthCalColor	Retrieves the color for the given portion of DTP control's child month calendar control.
getMonthCalFont	Gets the font that the DTP control's child month calendar control is currently using.
getMonthCalStyle	Retrieves the style keywords of the date and time picker's child month calendar control.
getRange	Gets the current minimum and maximum allowable times for a date and time picker (DTP) control.
setDateTime	Sets the date and time for the DTP control to that specified.
setFormat	Sets the display of a date and time picker (DTP) control based the specified format string.
setMonthCalColor	Sets the color for the specified portion of the DTP control's child month calendar control.
setMonthCalFont	Sets the font to be used by the date and time picker control's child month calendar control.
setMonthCalStyle	Sets the style for the DTP control's child month calendar control.
setRange	Sets the minimum and maximum allowable dates / times for the date time picker control.

12.4. newDateTimePicker (dialog object method)

`DateTimePicker` objects can not be instantiated by the programmer from Rexx code using a *new*() method. Rather, a date and time picker object is obtained by using the *newDateTimePicker* method of the *dialog* object. The syntax is:

```
>>-newDateTimePicker(--id--)---------------------><
```

12.5. createDateTimePicker (UserDialog method)

A date and time picker control can be added to the dialog template of a *UserDialog*) through the *createDateTimePicker* method. The basic syntax is:

```
>>-createDateTimePicker(-id-,--x-,--y-,--cx-,--cy-+---------+-+------------+--><
                                                  +-,--style-+ +-,--attrName-+
```

12.6. connectDateTimePickerEvent (dialog object method)

To connect event notifications from a DTP control use the *connectDateTimePickerEvent* method of the *dialog* object. The basic syntax is:

```
>>--connectDateTimePickerEvent(--id-,-evt--+---------+-+-----------+-)------><
                                           +-,--mName-+ +-,-wilReply-+
```

12.7. closeMonthCal

```
>>--closeMonthCal-------------------------------><
```

Closes the drop down month calendar control of the date time picker.

Arguments:

The method takes no arguments.

Return value:

The method always returns 0.

Remarks:

This method causes the date time picker to destroy the month calendar control and to send the DTN_CLOSEUP notification (the *CLOSEUP* event notification) that the drop-down has been closed.

Details

Requires Windows Vista or later.

12.8. getDateTime

```
>>--getDateTime---------------------------------><
```

Retrieves the currently selected date and time for the DTP and returns it as a **`DateTime`** object.

Arguments:

This method has no arguments.

Return value:

Returns a **`DateTime`** object representing the current selected date and time of the DTP control. May return the .nil object if the DTP control has the SHOWNONE style and is in the *no date* state. It is possible that an error may occur within the underlying DTP control itself. If this is detected, 0 is returned.

Remarks:

When the DTP has the SHOWNONE style, the control includes a check box allowing the user to indicate that *no* date is selected. In this case the **`.nil`** object is returned to indicate that no date is selected.

Details

Sets the *.systemErrorCode* if an error occurs within the underlying DTP control. The system error code is set this way by the ooDialog framework:

ERROR_INVALID_MESSAGE (1002)

The window cannot act on the sent message. **Meaning:** The underlying DTP control returned an error rather than the selected data and time.

12.9. getIdealSize

```
>>--getIdealSize(--size--)-----------------------><
```

Gets the size needed to display the date time picker control without clipping.

Arguments:

The single required argument is:

size [required] [in / out]

A *Size* object in which the ideal size is returned.

Return value:

Returns true always because the method always succeeds.

Details

Requires Windows Vista or later.

Raises syntax errors when incorrect usage is detected.

Example:

This example sets a longish format for a DTP control and then resizes the control to its ideal size:

```
::method initDialog
  expose dtp

  dtp = self~newDateTimePicker(IDC_DTP);

  format = 'dddd MMMM d, yyyy - h:mm tt'
  dtp~setFormat(format)

  time = .DateTime~fromISODate('2011-03-01T10:00:00.000000')
  dtp~setDateTime(time)

  ideal = .Size~new
  dtp~getIdealSize(ideal)

  dtp~resizeTo(ideal)
  ...
```

12.10. getInfo

```
>>--getInfo--------------------------------------><
```

Returns a **`Directory`** object with information about the date time picker.

Arguments:

This method takes no arguments.

Return value:

Returns a **`Directory`** object whose indexes contain information concerning the DTP control. The indexes of the **`Directory`** object will be:

CHECKRECT
: This index is a *Rect* object that contains the location of the check box allowing the user to have no date selected. This check box is only present when the DTP control has the SHOWNONE style. When the check box is not present, the coordinates of the rectangle will be all zeros.

CHECKSTATE
: A localized string describing the state of the check box allowing the user to have no date selected. This check box is only present when the DTP control has the SHOWNONE style. The operating system associates one or more states with any object at any given time.

  The string at index CHECKSTATE is the string the operating system returns to describe an object's state. Microsoft's documentation is a little vague on exactly what the string will be. However, from testing it appears the string will be *invisible* when the check box is not present. When the check box is present, it appears the string will be *checked* or *normal*.

BUTTONRECT
: A *Rect* object that contains the location of the button that shows the drop down calendar when the user clicks on it. (Or closes the month calendar if the user clicks on it when the month calendar is already open.) This button is not present if the DTP control has the UPDOWN style, in which case the coordinates of the rectangle will be all zeros.

BUTTONSTATE
: The BUTTONSTATE index is similar to the CHECKSTATE index. Its value is a localized string describing the state of the button for the drop down month calendar. When the DTP control has the UPDOWN style and the button is not present, the string is *invisible*. When the month calendar is showing the state of the button is *pressed*, when it is not showing the state will be either *normal* or *unavailable*. The state is *unavailable* if the DTP control has the SHOWNONE style and no date is currently selected.

EDIT
: This index will either be an *Edit* object, or **`.nil`**, depending on whether the DTP control currently has a child edit control or not. The DTP control will only have an edit control when the DTP control has the CANPARSE style. The CANPARSE style allows the user to edit within the client area of the control when they press the F2 key, or single click on the client area of the DTP control when the control already has the focus. The child edit control is only present when the user is actively editing.

DROPDOWN
: This index will either be a *MonthCalendar* object, or **`.nil`**, depending on whether the DTP control currently has a child month calendar control or not.

UPDOWN
: This index will either be an *UpDown* object, or **`.nil`**, depending on whether the DTP control currently has a child updown control or not. The up down control will only exist when the DTP control has the UPDOWN style. With the UPDOWN style, the normal drop down month calendar is replaced by an up down control.

Details

Requires Windows Vista or later.

Example:

This example connects the **`F3`** key to the dialog. When the user presses the **`F3`** key, the current DTP control information is displayed:

```
::method defineDialog

  self~createDateTimePicker(200,  10, 7, 280, 15, "BORDER SHORT SHOWNONE")
  ...

::method initDialog
  expose dateTime

  dateTime = self~newDateTimePicker(IDC_DTP_BIRTHDAY);
  dateTime~setFormat("XX 'is on: 'hh':'mm':'ss dddd MMM dd', 'yyyy XXX")

  self~connectKeyPress(onF3, .VK~f3)
  ...

::method onF3 unguarded
  expose dateTime

  info = dateTime~getInfo
  say 'CheckRect:   ' info~checkRect
  say 'CheckState:  ' info~checkState
  say 'ButtonRect:  ' info~buttonRect
  say 'ButtonState:' info~buttonState
  say 'DropDown:    ' info~dropDown
  say 'Edit:        ' info~edit
  say 'UpDown:      ' info~upDown
  say

/* As the user presses F3 at various times the output could be:

CheckRect:   a Rect (2, 2, 22, 22)
CheckState:  checked
ButtonRect:  a Rect (386, 0, 420, 24)
ButtonState: normal
DropDown:    The NIL object
Edit:        The NIL object
UpDown:      The NIL object

CheckRect:   a Rect (2, 2, 22, 22)
CheckState:  checked
ButtonRect:  a Rect (386, 0, 420, 24)
ButtonState: pressed
DropDown:    a MonthCalendar
Edit:        The NIL object
UpDown:      The NIL object

CheckRect:   a Rect (2, 2, 22, 22)
CheckState:  normal
ButtonRect:  a Rect (386, 0, 420, 24)
ButtonState: unavailable
DropDown:    The NIL object
Edit:        The NIL object
UpDown:      The NIL object

*/
```

12.11. getMonthCal

```
>>--getMonthCal----------------------------------><
```

Returns the *MonthCalendar* object for a date and time picker's (DTP) child month calendar control.

Arguments:

This method takes no arguments.

Return value:

A **`MonthCalendar`** object for the underlying child month calendar, if the child control exists, otherwise the **`.nil`** object.

Remarks:

Note: The following information mostly applies to Windows XP and earlier. For Windows Vista and later the date and time picker control has the *setMonthCalStyle* and *getMonthCalStyle* methods to directly manipulate the month calendar control's styles.

Obtaining the month calendar control allows the programmer to change its style or appearance. For instance the *setFirstDayOfWeek* day of the week could be changed or the *Go To Today* could be removed by setting the NOTODAY style.

DTP controls do not create a static month calendar control. Rather, they create the month calendar when needed and destroy the month calendar as soon as it is not needed. The *DROPDOWN* notification is sent after the DTP creates the month calendar and the *CLOSEUP* notification is sent when the month calendar is destroyed.

After the underlying month calendar control is destroyed, Rexx month calendar object is no longer valid.

Example:

This example will work on any version of Windows that ooDialog supports. It gets the month calendar for the DTP control and changes the first day of the week to Friday. The *Today* selection is also removed. This is an example of a *DROPDOWN* event handler. Note that no reference is kept for the month calendar. The object will be invalid as soon as the user closes the drop down:

```
::method onDropDown unguarded
  use arg idFrom, hwndFrom

  dt = self~newDateTimePicker(IDC_DTP);
  monthCal = dt~getMonthCal
  monthCal~setFirstDayOfWeek('Friday')
  monthCal~addStyle("NOTODAY")

  return 0
```

12.12. getMonthCalColor

```
>>--getMonthCalColor(--calendarPart--)----------><
```

Retrieves the color for the given portion of DTP control's child month calendar control.

Arguments:

The single argument is:

calendarPart [required]

A single keyword specifying which part of the calendar to get the color for, case is not significant. The valid keywords are:

BACKGROUND	TITLEBK
MONTHBK	TITLETEXT

TEXT TRAILINGTEXT

BACKGROUND
: Retrieves the background color displayed between months.

MONTHBK
: Retrieves the background color displayed within the month.

TEXT
: Retrieves the color used to display text within a month.

TITLEBK
: Retrieves the background color displayed in the calendar's title.

TITLETEXT
: Retrieves the color used to display text within the calendar's title.

TRAILINGTEXT
: Retrieves the color used to display header day and trailing day text. Header and trailing days are the days from the previous and following months that appear on the current month calendar.

Return value:

On success, returns a COLORREF representing the color setting for the part of the month calendar control specified. returns CLR_INVALID on error.

Remarks:

This method is essentially the same as the *getColor* method of the *MonthCalendar* class, except that DTP control is asked to get the color from its child month calendar.

The *Image* class has a number of convenience methods for working with COLORREFs, including the *colorRef* method that explains what a COLORREF is.

The *colorRef* method can be used to test the return for error. I.e.: **`.Image~colorRef(CLR_INVALID)`**. (An error is not very likely.)

Note that many of the underlying dialog controls ignore their color settings when visual themes are enabled, unless the theme is Windows Classic. This is the case for the month calendar control.

Details

Raises syntax errors when incorrect arguments are detected.

12.13. getMonthCalFont

```
>>--getMonthCalFont------------------------------><
```

Gets the font that the DTP control's child month calendar control is currently using.

Arguments:

There are no arguments for this method.

Return value:

The return is a *handle* to the font, the same return as the *getFont* or *createFontEx* methods of the *dialog* object would return.

Remarks:

Note that many of the underlying dialog controls ignore their font and color settings when visual themes are enabled, unless the theme is Windows Classic. This is the case for the month calendar control.

12.14. getMonthCalStyle

```
>>--getMonthCalStyle----------------------------><
```

Retrieves the style keywords of the date and time picker's child month calendar control.

Arguments:

This method does not have any arguments.

Return value:

A list of *MonthCalendar* style keywords separated by spaces. The list may be empty, otherwise it will contain one or more of the following keywords.

DAYSTATE	NOCIRCLE	SHORTDAYS
MULTI	WEEKNUMBERS	NOSELCHANGE
NOTODAY	NOTRAILING	

DAYSTATE

The month calendar will send the GETDAYSTATE *event* notification to request information about which days should be displayed in bold. For the DTP control's child month calendar, this style is not of much interest, and not of much use, to the ooDialog programmer. The ooDialog can not connect an event handler to the child month calendar.

MULTI

This style allows the user to select a range of dates within the control. By default, the maximum range is one week. The programmer changes the maximum range that can be selected by using the *setMaxSelection* method. Again, this style is not relevant for the DTP control's child month calendar control. The DTP control only supports a specific date, a range of dates is not supported.

NOTODAY

The month calendar control will not display the *today* date at the bottom of the control.

NOCIRCLE

The month calendar control will not circle the *today* date at the bottom of the control.

WEEKNUMBERS

The month calendar displays week numbers (1-52) to the left of each row of days. Week 1 is defined by Microsoft as the first week that contains at least four days.

NOTRAILING

Windows Vista or later only. This style disables displaying the dates from the previous / next month in the current calendar.

SHORTDAYS

Windows Vista or later only. With this style, the month calendar uses the shortest day name for the label of the day of the week column header.

NOSELCHANGE

Windows Vista or later only. With this style the selection does not change when the user navigates to the next or previous month in the calendar. This style is needed for the user to be able to select a range greater than what is currently displayed. This style is also not relevant for the same reasons as given for the MULTI style.

Details

Requires Windows Vista or later.

Example:

This example shows a simple exploratory program that could be written to see what the default style for the DTP control's child month calendar is. Note that, in the author's experience, the child month calendar does not have any styles set.

For an expanded version of this example, see the *example* for the *setMonthCalStyle* method.

```
/*
 * Simple test program to display the DTP control's month calendar's initial
 * style.
 */

  if \ .OS~isAtLeastVista then do
    say "Sorry this example program is for Windows Vista or later."
    return 99
  end

  .application~useGlobalConstDir('O')
  .constDir[IDC_DTP] = 200

  dlg = .TestDialog~new
  dlg~execute("SHOWTOP")

return 0

::requires "ooDialog.cls"

::class 'TestDialog' subclass UserDialog

::method init

  forward class (super) continue
  self~createCenter(200, 100, 'Dialog', , , 'MS Shell Dlg 2', 8)

::method defineDialog

  self~createDateTimePicker(IDC_DTP,  10, 7, 180, 15, "SHORT")
  self~createPushButton(IDOK, 140, 74, 50, 14, "DEFAULT", "Ok")

  self~connectDateTimePickerEvent(IDC_DTP, "DROPDOWN", onDropDown)

::method onDropDown unguarded
  use arg idFrom, hwndFrom

  dtp = self~newDateTimePicker(IDC_DTP)
  say "Style:" dtp~getMonthCalStyle
  return 0

/* Output will be: */
```

12.15. getRange

```
>>--getRange(--range--)--------------------------><
```

Gets the current minimum and maximum allowable times for a date and time picker (DTP) control.

Arguments:

The single required argument is:

range [required] [in / out]

An **`Array`** object in which the minimum and maximum times are returned as **`DateTime`** objects. The minimum time will be at index 1 and the maximum at index 2. If either index is set to zero, then no corresponding limit is set for the date time picker control.

Return value:

The return will be a keyword indicating the result. See the Remarks section for possible keywords.

Remarks:

The returned keyword will be exactly one of: **`none, min, max, both, error`** describing which limit(s) are set. The *error* keyword would indicate an error occurred with the DTP control, but that is very unlikely.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example shows a method designed to display the current range of the DTP control:

```
::method showRange
  expose dtp

  min_max = .array~new(2)
  keyword = dtp~getRange(min_max)

  select
    when keyword == 'none' then do
      say "There are no minimum or maximum times set for the DTP control."
    end

    when keyword == 'min' then do
      say "The minimum time for the DTP control is:" min_max[1]'.'
      say 'There is no maximum time set.'
    end

    when keyword == 'max' then do
      say "The maximum time for the DTP control is:" min_max[2]'.'
      say 'There is no minimum time set.'
    end

    when keyword == 'both' then do
      say "The minimum time for the DTP control is:" min_max[1]'.'
      say "The maximum time for the DTP control is:" min_max[2]'.'
    end

    when keyword == 'error' then do
      say 'An unexplained error occurred.'
    end

    otherwise do
      -- This is not possible
      nop
    end
  end
  -- End select
```

```
/*  Output might for instance be:

The minimum time for the DTP control is: 2011-01-01T00:00:00.000000.
The maximum time for the DTP control is: 2011-12-31T11:59:59.999000.

*/
```

12.16. setDateTime

```
>>--setDateTime(--dateTime--)--------------------><
```

Sets the date and time for the DTP control to that specified.

Arguments:

The single required argument is:

dateTime [required]

A **`DateTime`** object that specifies the date and time to set the DTP to. However, if the DTP control has the SHOWNONE style, the *dateTime* argument can be the **`.nil`** object to set the DTP control to the *no date* state.

Return value:

Returns 0, always.

Remarks:

When the DTP control has the SHOWNONE style, it is possible to have no date currently selected. The control displays a check box that is automatically selected whenever a date is picked or entered. If the check box is subsequently deselected, the application cannot retrieve the date from the control because, in essence, the control has no date. The *no date* state can be set with the *setDateTime* method and queried by using the *getDateTime* method.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example sets the date and time of the DTP control to March 1st, 2011 at 10:00 am:

```
::method initDialog
  expose dtp

  dtp = self~newDateTimePicker(IDC_DTP);

  time = .DateTime~fromISODate('2011-03-01T10:00:00.000000')
  dtp~setDateTime(time)
```

12.17. setFormat

```
>>--setFormat(--format--)------------------------><
```

Sets the display of a date and time picker (DTP) control based the specified *format* string.

Arguments:

The single required argument is:

format [required]

A *format* string that dictates how the DTP control formats the data and time in its display.

Return value:

Returns true on success, false on error.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example creates a format string and then sets the DTP to use it:

```
::method initDialog
  expose dtp

  dtp = self~newDateTimePicker(IDC_DTP);

  format = 'dddd MMMM d, yyyy - h:mm tt'
  dtp~setFormat(format)

/* Display will look like this, for example, depending on exact
   date and time:

Monday October 24, 2011 - 11:52 AM

*/
```

12.18. setMonthCalColor

```
>>--setMonthCalColor(--which--,--color--)--------><
```

Sets the color for the specified portion of the DTP control's child month calendar control.

Arguments:

The arguments are:

which [required]

Specifies which part of the calendar will have its color set. Exactly one of the following keywords must be used, case is not significant:

BACKGROUND	TEXT	TITLETEXT
MONTHBK	TITLEBK	TRAILINGTEXT

BACKGROUND

Set the background color displayed between months.

MONTHBK

Set the background color displayed within the month.

TEXT

Set the color used to display text within a month.

TITLEBK

Set the background color displayed in the calendar's title.

TITLETEXT

Set the color used to display text within the calendar's title.

TRAILINGTEXT

Set the color used to display header day and trailing day text. Header and trailing days are the days from the previous and following months that appear on the current month calendar.

color [required]

The color, specified as a COLORREF, to set the specified calendar part.

Return value:

On success, returns the previous color, as a COLORREF, for the part of the month calendar control specified. Returns CLR_INVALID on error.

Remarks:

The *Image* class has a number of convenience methods for working with COLORREFs, including the *colorRef* method that explains what a COLORREF is.

The *colorRef* method can be used to test the return for error. I.e.: **`.Image~colorRef(CLR_INVALID)`**. (An error is not very likely.)

Note that many of the underlying dialog controls ignore their font and color settings when visual themes are enabled, unless the theme is Windows Classic. This is the case for the month calendar control.

Details

Raises syntax errors when incorrect arguments are detected.

12.19. setMonthCalFont

```
>>--setMonthCalFont(--newFont--+-----------+--)--><
                               +-,-redraw--+
```

Sets the font to be used by the date and time picker control's child month calendar control.

Arguments:

The arguments are:

newFont [required]

The new font for the child month calendar. One way to obtain a font is though the *createFontEx* method.

redraw [optional]

Specifies whether the month calendar control should redraw itself immediately upon setting the font. The default is true, which causes the control to redraw itself immediately.

Return value:

xx

Remarks:

Note that many of the underlying dialog controls ignore their font and color settings when visual themes are enabled, unless the theme is Windows Classic. This is the case for the month calendar control.

12.20. setMonthCalStyle

```
>>--setMonthCalStyle(--style--)------------------><
```

Sets the style for the DTP control's child month calendar control.

Arguments:

The single required argument is:

style [required]

A list of *MonthCalendar* style keywords separated by spaces. The list may be empty. This will revert the month calendar to its default style. Otherwise, it can contain one or more of the following keywords.

DAYSTATE	NOCIRCLE	SHORTDAYS
MULTI	WEEKNUMBERS	NOSELCHANGE
NOTODAY	NOTRAILING	

DAYSTATE

The month calendar will send the GETDAYSTATE *event* notification to request information about which days should be displayed in bold. For the DTP control's child month calendar, this style is not of much interest, and not of much use, to the ooDialog programmer. The ooDialog can not connect an event handler to the child month calendar.

MULTI

This style allows the user to select a range of dates within the control. By default, the maximum range is one week. The programmer changes the maximum range that can be selected by using the *setMaxSelection* method. Again, this style is not relevant for the DTP control's child month calendar control. The DTP control only supports a specific date, a range of dates is not supported.

NOTODAY

The month calendar control will not display the *today* date at the bottom of the control.

NOCIRCLE

The month calendar control will not circle the *today* date at the bottom of the control.

WEEKNUMBERS

The month calendar displays week numbers (1-52) to the left of each row of days. Week 1 is defined by Microsoft as the first week that contains at least four days.

NOTRAILING

Requires Windows Vista or later. This style disables displaying the dates from the previous / next month in the current calendar.

SHORTDAYS

Requires Windows Vista or later. With this style, the month calendar uses the shortest day name for the label of the day of the week column header.

NOSELCHANGE

Requires Windows Vista or later. With this style the selection does not change when the user navigates to the next or previous month in the calendar. This style is needed for the user to be able to select a range greater than what is currently displayed. This style is also not relevant for the same reasons as given for the MULTI style.

Return value:

The return value is a list of style keywords, (see the *style* argument above,) defining the style the child month calendar had previously.

Example:

This example is a stand-alone program. It is a slightly expanded version of the *getMonthCalStyle example*.

```
/*
 * Simple test program to set and show the DTP control's month calendar's
 * initial style.
 */

  .application~useGlobalConstDir('O')
  .constDir[IDC_DTP] = 200

  dlg = .TestDialog~new
  dlg~execute("SHOWTOP")

return 0

::requires "ooDialog.cls"

::class 'TestDialog' subclass UserDialog

::method init

  forward class (super) continue
  self~createCenter(200, 100, 'Dialog', , , 'MS Shell Dlg 2', 8)

::method defineDialog

  self~createDateTimePicker(IDC_DTP,  10, 7, 180, 15, "SHORT")
  self~createPushButton(IDOK, 140, 74, 50, 14, "DEFAULT", "Ok")

  self~connectDateTimePickerEvent(IDC_DTP, "DROPDOWN", onDropDown)
  self~connectDateTimePickerEvent(IDC_DTP, "CLOSEUP", onDropDown)

::method initDialog
  expose dtp

  dtp = self~newDateTimePicker(IDC_DTP);

  if .OS~isAtLeastVista then do
    dtp~setMonthCalStyle("NOCIRCLE NOTRAILING")
  end

::method onDropDown unguarded
  expose dtp
  use arg idFrom, hwndFrom

  if \ .OS~isAtLeastVista then do
    mc = dtp~getMonthCal
    mc~replaceStyle('DAYSTATE MULTI WEEKNUMBERS', "NOCIRCLE NOTODAY")
  end

  if .OS~isAtLeastVista then do
    say "Style:" dtp~getMonthCalStyle
  end
  else do
    say "Style (Windows 2000 or XP):" mc~getStyle
  end
  return 0

/* Output on Vista or later will be:

Style: NOCIRCLE NOTRAILING

Output on Windows 2000 or XP:
```

```
Style (Windows 2000 or XP): NOCIRCLE NOTODAY

*/
```

12.21. setRange

```
>>--setRange(--dateTimes--)----------------------><
```

Sets the minimum and maximum allowable dates / times for the date time picker control.

Arguments:

The single required argument is:

dateTimes [required]

An array of **`DateTime`** objects used to set the minimum and maximum dates. The **`DateTime`** object at index 1 sets the minimum date and the **`DateTime`** object at index 2 sets the maximum date.

Return value:

True on success, otherwise false.

Remarks:

The array must contain at least one of the indexes. If it contains neither, an exception is raised. If one of the array indexes is empty, then the corresponding date is not set.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example restricts the range of the DTP control to the year 2011:

```
::method initDialog
  expose dtp

  dtp = self~newDateTimePicker(IDC_DTP);

  min = .DateTime~fromUsaDate('01/01/11')
  max = .DateTime~fromISODate('2011-12-31T23:59:59.999999')

  dtp~setRange(.array~of(min, max))
```

Edit Controls

An edit control is a rectangular window that allows the user to enter and edit text using the keyboard. Edit controls have two line styles, single-line and multiline.

When the edit control has the focus it displays a blinking line called the caret that marks the insertion point in the control. The insertion point can be changed using the keyboard or the mouse. Text can be selected, also using the keyboard or mouse.

The **`Edit`** class provides methods to work with and manipulate the underlying Windows edit dialog control which it represents. It is a concrete subclass of the dialog *control* object and therefore has all methods of the of the dialog control object.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with edit controls:

Instantiation:

Use the *newEdit* method of the *dialog* object to retrieve a new Edit object.

Dynamic Definition:

To dynamically define an edit in a *UserDialog* class, use the *createEdit* method.

Event Notification

To connect the *event* notifications sent by the underlying edit control to a method in the Rexx dialog object use the *connectEditEvent*() method.

13.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with Edit objects, including the pertinent methods from other classes:

Table 13.1. Important Edit Methods

Method	Description
Useful	**External Methods**
newEdit	Returns a new **`Edit`** object for the edit control with the specified ID.
createEdit	Creates an edit control in the dialog template of a *UserDialog*
connectEditEvent	Connects an edit control event notification to a method in the Rexx dialog object
Attributes	**Attributes**
passwordChar	Set or get the password character for an edit control with the PASSWORD style.
Instance Methods	**Instance Methods**
addStyle	Adds one or more edit control styles to the edit control.
canUndo	Determines if there are any actions in the undo queue of the edit control.
clearText	Deletes the currently selected text from the edit control.
copyText	Copies the currently selected text to the clipboard in text format.
cutText	Deletes the currently selected text in the edit control and copies it to the clipboard.
disableInternalResize	Prevents the edit control from internally resizing itself when its owner dialog is resized.

Method	Description
ensureCaretVisibility	Scrolls the edit control until the caret (cursor) is visible.
firstVisibleLine	Retrieves the index of the first visible line, or the first visible character, in an edit control.
getCue	Retrieves the cue text, or the empty string if there is no cue set.
getLine	Retrieves the text of the specified line.
getMargins	Returns the width of the left and right margins of the edit control in a **`Directory`** object.
getRect	Returns the edit control's current formatting rectangle as a **`.Rect`** object
getStyle	Returns the edit control's current style as a blank delimited list of style keywords.
hideBalloon	Hides the balloon information window displayed by the *showBalloon* method.
isGrandchild	Notifies the ooDialog framework that this edit control is a grandchild of the dialog and configures the underlying edit control to send four event notifications to the dialog, rather than its direct parent.
isModified	Determines if the text in the edit control has been modified.
isSingleLine	Determines if this edit control is a single line or a multi-line edit control.
lineFromIndex	Gets the line index of the line that contains the specified character index.
lineIndex	Determines the one-based character index of the first character in the line specified.
lineLength	Returns the length, in characters, of the specified line in an edit control.
lines	Determines the number of lines in a multiline edit control.
lineScroll	Scrolls the text horizontally and vertically in a multiline edit control.
margins	Returns the left and right margins of the edit control as a **`String`** object.
noContextMenu	Removes the edit control's internal context menu.
pasteText	Copies the content of the clipboard to the edit control at the current caret position.
removeStyle	Removes one or more edit control styles from the edit control.
replaceSelText	Replaces the text selection in an edit control with the specified text.
replaceStyle	Removes and adds one or more edit control styles in one operation.
scrollCommand	Scrolls the text vertically in a multiline edit control.
select	Used to change the text selection and / or the position of the cursor
selected	Returns a **`String`** object containing the start and end of the text selection.
selection	Returns a **`Directory`** object containing the start and end of the text selection.
setCue	Sets the text for the *cue* banner.
setLimit	Sets the maximum number of characters the user can type in to the edit control.
setMargins	Sets the left and right margins of the edit control as specified.
setModified	Sets the modified flag of an edit control to the state specified, **`.true`** or **`.false`**.

Method	Description
setReadOnly	Adds or removes the read only style for the edit control.
setRect	Sets the edit control's current formatting rectangle using a `.Rect` object
setTabStops	Sets the tab stops for text copied into a multi-line edit control.
showBalloon	Displays an informational window, a balloon tip, for the edit control.
tab =	An alias for the *setTabStops* method.
undo	Undoes the last edit control operation in the edit control's undo queue.
wantReturn	Connects the *Enter* keypress to a method in the Rexx dialog and prevents the edit control from sending the keypress to the dialog manager, which closes the dialog.

13.2. newEdit (dialog object method)

Edit objects can not be instantiated by the programmer from Rexx code. Rather an Edit object is obtained by using the *newEdit* method of the dialog *dialog* object. The syntax is:

```
>>--newEdit(--id--)------------------------------><
```

13.3. createEdit (UserDialog method)

An edit control can be added to the dialog template for a *UserDialog* dialog through the *createEdit* method. The basic syntax is:

```
>>--createEdit(-id-,--x-,--y-,--cx-,--cy-+---------+--+----------------+--)---><
                                         +-,-style-+  +-,-attributeName-+
```

13.4. connectEditEvent (dialog object method)

To connect event notifications from an edit control use the *connectEditEvent* method of the *dialog* object. The basic syntax is:

```
>>--connectEditEvent(--id--,--event--+----------------+--)-----><
                                     +--,-methodName--+
```

13.5. passwordChar (Attribute)

```
>>--passwordChar---------------------------------><

>>--passwordChar = varName-----------------------><
```

The *passwordChar* attribute reflects the password character for a password edit control.

passwordChar get:

Returns the current password character for the edit control, if it is set, otherwise the empty string.

passwordChar set:

Sets the password character for the edit control.

Remarks:

The password character only has meaning if the edit control has the PASSWORD style, otherwise it is ignored. Multiline edit controls don't support the PASSWORD style or getting and setting the password character.

When the operating system creates the underlying edit control, if it has the PASSWORD style, it sets the password character to the default. This is an asterisk for earlier versions of Common Control *Library*, a solid circle for Comctl32 version 6.0 and later.

13.6. addStyle

```
>>--addStyle(--style--)--------------------------><
```

Adds one or more edit control styles to the edit control.

Arguments:

The single argument is:

style [required]

A list of 1 or more of the following keywords separated by spaces, case is not significant:

UPPER	WANTRETURN	GROUP
LOWER	OEM	
NUMBER	TAB	

UPPER

Converts all characters to uppercase as they are typed into the edit control.

LOWER

Converts all characters to lowercase as they are typed into the edit control.

NUMBER

Only allows digits to be typed into the edit control. If the user types an non-digit character an information balloon is displayed informing the user that only digits are allowed. This style does not prevent the user from pasting non-digits into the control.

WANTRETURN

This keyword only effects multi-line edit controls. With this style when a user hits the enter key the line break characters are inserted into the text buffer. Without the style, when the user hits enter the default push button of the dialog is pressed.

OEM

Converts text entered in the edit control from the Windows character set to the OEM character set and then back again to the Windows character set. This ensures proper character conversion in certain circumstances. Microsoft states that this style is most useful when edit controls contain file names that will be used with file systems that do not support Unicode. For further details see the Windows *documentation* provided by Microsoft.

TAB

Adds the *tabstop* style.

GROUP

Adds the *group* style.

Return value:

The positive numeric value of the edit control's previous style.

It is possible, but very unlikely, that a negative number is returned. See the remarks.

Remarks:

This method only works with the window *styles* that can be changed after the edit control has been created. Not all window styles can be changed after the control is created.

If an operating system error occurs, this method returns the negated value of the system error code. However, this method also sets **`.systemErrorCode`** if there is an error.

The negative number return is a hold over from earlier versions of ooDialog before **`.systemErrorCode`** was introduced.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example shows the syntax to add the number, tabstop, and group styles to an edit control.

```
ec = dlg~newEdit(IDC_EDIT_PARTNUMBERS)
ec~addStyle("NUMBER TAB GROUP")
```

13.7. canUndo

```
>>--canUndo-------------------------------------><
```

Determines if there are any actions in the undo queue of the edit control.

Arguments:

There are no arguments for this method.

Return value:

The return value is **`.true`** if there are actions in the undo queue, otherwise **`.false`**.

Remarks:

When the undo queue is not empty, the *undo* method can be used to undo the most recent operation.

13.8. clearText

```
>>--clearText-----------------------------------><
```

Deletes (clears) the current selected text, if any, from the edit control

Arguments:

There are no arguments for this method.

Return value:

This method always returns 0.

Remarks:

When the *clearText* method is used to delete the selected text, the deletion can be undone using the *undo* method.

13.9. copyText

```
>>--copyText-------------------------------------><
```

Copies the currently selected text in the edit control to the clipboard, in text format.

Arguments:

There are no arguments to this method.

Return value:

This method always returns 0.

13.10. cutText

```
>>--cutText(--+--------+--)----------------------><
              +--type--+
```

Deletes (cuts) the currently selected text, if any, in the edit control and copies the deleted text to the clipboard in text format.

Arguments:

There are no arguments to this method.

Return value:

This method always returns 0.

Remarks:

The deletion performed by the *cutText* method can be undone by using the *undo* method. To delete the currently selected text without placing that text on the clipboard, use the *clearText* method.

13.11. disableInternalResize

```
>>--disableInternalResize(--+-----------+--)-----><
                            +--disable--+
```

The edit control, internally, resizes itself in response to its parent dialog being resized. There are times when this behavior may not be desired. The *disableInternalResize* method can be used to disable the behavior.

Arguments:

The single argument is:

disable [optional]

.true or **.false**. If true, disable the edit control's internal resizing. If false, re-enable the behavior, if it has been disabled. The default is **.true**

Return value:

True on success, false on failure.

Remarks:

If the programmer is resizing the edit control himself in response to the resizing of the dialog, the edit control's internal resizing may get in the way. That behavior of the edit control can be suppressed by using the *disableInternalResize* method. The programmer can disable / re-enable the internal resizing of the edit control as often and whenever he wants.

Details

Requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version* method to determine the current version of the library. A syntax error is raised if this method is invoked on an operating system where it is not supported.

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*. The following codes may be set. They are set by the ooDialog framework rather than the operating system and are used to indicate error conditions:

ERROR_NOT_SUPPORTED (50)

The request is not supported. **Meaning:** The method was asked to disable internal resizing when it was already disabled, or asked to re-enable the behavior when it is not disabled to begin with.

ERROR_SIGNAL_REFUSED (156)

The recipient process has refused the signal. **Meaning:** The method tried to disable or re-enable the internal resizing, but failed.

Example:

The **samples/oodialog/dlgAreaUDemoTwo.rex** example program shows a resizable dialog. In this example, the program waits until the user is finished sizing the dialog before it updates the size and position of the dialog controls. The eliminates the flicker seen when the dialog controls are resized continually as the dialog is resized. In this program, the edit control has its internal resizing disabled, since it defeats the purpose of the program:

```
::method initDialog

  self~newEdit(IDC_EDIT)~disableInternalResize
```

13.12. ensureCaretVisibility

```
>>--ensureCaretVisibility------------------------><
```

Scrolls the position of the caret so that it is visible in the edit control.

Arguments:

There are no arguments for this method.

Return value:

This method always returns 0.

Details

Raises syntax errors when incorrect arguments are detected.

13.13. firstVisibleLine

```
>>--firstVisibleLine-----------------------------><
```

Retrieves the index of the first visible line in a multiline edit control or the first visible character in a single-line edit control.

Arguments:

This method takes no arguments.

Return value:

The index of the first visible line or first visible character depending on the type of the edit control.

Remarks

As in all methods of the **`Edit`** class, both the line index and the character index are one-based indexes.

Example:

This example ...

```
edit = self~newEdit(IDC_EDIT)

index = edit~firstVisibleLine
if edit~isSingleLine then do
  say 'The first visible character in the edit control is at index:' index
else
  say 'The first visible line in the edit control is at index:' index
```

13.14. getCue

```
>>--getCue---------------------------------------><
```

Retrieves the cue text, or the empty string if there is no cue set. The cue text is set with the *setCue* method.

Arguments:

There are no arguments for this method.

Return value:

Returns the text of the cue banner on success. The empty string is returned if the cue is not set, or on any error.

Remarks:

Cue text provides a visual hint to the user as to the purpose of the edit control. For example the text of an edit control that is used to begin a search might have the cue text: *Enter search here*. The cue is displayed in gray text. Once some text is entered into the edit box, the cue no longer displays.

Normally when the user clicks the on the edit control, the cue text goes away. However on Windows Vista and later this behavior can be changed so that the cue text remains until the user actually types some characters in the edit control.

It is not possible to set a cue on a multiline edit control.

On Windows XP the *getCue* method does not work, it will always return the empty string. This is the operating system behavior, it has nothing to do with ooDialog. On Windows Vista and later it correctly returns the cue text.

Details

Requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version*() method to determine the current version of the library. A syntax error is raised if this method is invoked on an operating system where it is not supported.

13.15. getLine

```
>>--getLine(--lineIndex--)-----------------------><
```

Retrieves the text of the specified line.

Arguments:

The single required arguments is:

lineIndex [required]

The one-based index of the line whose text is desired. This argument can be 0 to specify the current line.

Return value:

The text of the line specified on success. On error, the empty string is returned. Remember that the line specified may actually have no text. The programmer can check **`.systemErrorCode`** to determine if the return is an error indication or not.

Remarks:

The carriage return and line-feed characters are not included in the returned string. The current line is the line with the caret in it.

In earlier releases of ooDialog the *getLine* method required an optional, second, argument if the length of the text in a line was greater than 255 characters. This argument is no longer required, or used, in any circumstances. For backwards compatibility, the argument is accepted, but completely ignored. The complete line of text is returned no matte how long it may be.

Details

Sets the *.systemErrorCode*. The following error code is set by the ooDialog framework on error.

ERROR_NOT_SUPPORTED (50)

The request is not supported. **Meaning:** Indicate that *lineIndex* is not valid, it is greater than the number of lines in the edit control.

Raises syntax errors when incorrect arguments are detected.

Example:

This example prints to the screen the text in the edit control with the *symbolic* ID of **`IDC_EDIT_COPYRIGHT`**

```
  ec = self~newEdit(IDC_EDIT_COPYRIGHT)
```

```
do i = 1 to ec~lines
   say ec~getLine(i)
end
```

13.16. getMargins

```
>>--getMargins-----------------------------------><
```

Returns the width of the left and right margins of the edit control in a **`Directory`** object.

Arguments:

There are no arguments to this method.

Return value:

The return is a **`Directory`** object. The directory has the width of the left margin in the *left* index and the width of the right margin in the *right* index. Both values are in pixels.

Example:

This example is from a hypothetical application where the user can increase or decrease the margins in an edit control by clicking on a '+' or a '-' minus button.

```
::method onButtonClick unguarded
  expose editControl
  use arg info, handle

  id = .DlgUtil~loWord(info)
  margins = editControl~getMargins

  if id == self~constDir[IDC_PB_PLUS] then do
    if margins~left < 100 then margins~left += 2
    if margins~right < 100 then margins~right += 2
  end
  else if id == self~constDir[IDC_PB_MINUS] then do
    if margins~left > 2 then margins~left -= 2
    if margins~right > 2 then margins~right -= 2
  end
  else do
    return 0
  end

  editControl~setMargins(margins~left, margins~right)
  return 0
```

13.17. getRect

```
>>--getRect--------------------------------------><
```

Returns the current formatting rectangle of the edit control.

Arguments:

These method uses no arguments.

Return value:

The return from the *getRect* method is a **`.Rect`** object containing the coordinates for the formatting rectangle.

Remarks:

The visibility of the text in an edit control is dependent on its window rectangle and its formatting rectangle. The window rectangle is the client area of the window containing the edit control. The formatting rectangle is an artificial construct maintained by the operating system and used to format the text. The formatting rectangle can be set to be larger than the window rectangle, which will limit what can be seen of the text. Or the formatting rectangle can be made smaller than the window rectangle, which will create more white space around the text.

The coordinates of an edit control's formatting rectangle can be set through the *setRect* method.

Example:

This example prints the formatting rectangle of an edit control:

```
::method showFormattingRect private
  use strict arg editCtrl

  fRect = editCtrl~getRect
  say "Formatting rectangle: ("fRect~left',' fRect~top',' fRect~right',' fRect~bottom')'

/* Output might be for instance:

   Formatting rectangle: (2, 1, 562, 421)

*/
```

13.18. getStyle

```
>>--getStyle-------------------------------------><
```

Returns the edit control's current style as a blank delimited list of style keywords.

Arguments:

This method takes no arguments.

Return value:

A string consisting of blank delimited style keywords. The possible keywords are:

VISIBLE	VSCROLL	UPPER
HIDDEN	PASSWORD	LOWER
TAB	MULTILINE	RIGHT
NOTAB	AUTOSCROLLH	CENTER
DISABLED	AUTOSCROLLV	OEM
ENABLED	READONLY	LEFT
GROUP	WANTRETURN	KEEPSELECTION
HSCROLL	NUMBER	

The meaning of these style keywords is documented in the *createEdit* method.

It is possible, but very unlikely, that a negative number is returned. See the remarks.

Remarks:

If an operating system error occurs, this method returns the negated value of the system error code. However, this method also sets **`.systemErrorCode`** if there is an error.

The negative number return is a hold over from earlier versions of ooDialog before **`.systemErrorCode`** was introduced.

Details

Sets the *.systemErrorCode*.

Example:

This example queries the edit control's current style. If it has the NUMBER only style, it is removed.

```
ec = dlg~newEdit(IDC_EDIT)

style = editControl~getStyle
if style~wordPos("NUMBER") <> 0 then
  ec~removeStyle("NUMBER")
...
```

13.19. hideBalloon

```
>>--hideBalloon---------------------------------><
```

Hides the balloon information window displayed by the *showBalloon* method.

Arguments:

This method takes no arguments.

Return value:

Returns 0 on success, 1 if the method fails.

Remarks:

The operating system will dismiss the balloon window automatically after 10 seconds with no programmer intervention. The *hideBalloon* method can be used to dismiss the ballon before the 10 seconds is up.

Details

Raises syntax errors for incorrect usage.

Requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version*() method to determine the current version of the library. A syntax error is raised if this method is invoked on an operating system where it is not supported.

Example:

For an example see the showBalloon *example*.

13.20. isGrandchild

```
>>--isGrandchild(--+------------+--+------------+--)------------><
                   +-,-mthName--+  +-,-wantTab--+
```

Notifies the ooDialog framework that this edit control is a grandchild of the dialog and configures the underlying edit control to send four event notifications to the dialog, rather than its direct parent. The edit control must be a single-line edit control.

The four events that the grandchild notifies the Rexx dialog of are the RETURN, ESCAPE, TAB keydown events, and the KILLFOCUS event. See the Remarks section for more details.

Arguments:

The arguments are:

mthName [optional]
: The name of the method in the Rexx dialog that is connected to the four event notifications sent by the grandchild. If this argument is omitted, the default method name is *onEditGrandchildEvent*.

wantTab [optional]
: By default, the TAB keydown notification is not sent to the dialog. If the *wantTab* argument is true then the *mthName* method is also invoked for the TAB keydown event. The default for this argument is false and the TAB keydown event does not cause the method to be invoked.

Return value:

Returns true on success, false on error.

Remarks:

Normally dialog controls are child windows of the dialog window. Historically, in older versions of ooDialog, this was always the case. However, since ooDialog 4.2.0, there have begun to be some cases where the programmer can gain access to dialog controls that are not the direct child of the dialog. The setParent method can be used to change the parent window of a dialog control to a parent window other than the dilaog window. When the other window is a dialog control, this makes the control window a grandchild of the dialog. Some dialog controls create their own child dialog controls and enhancements to ooDialog have begun to give the Rexx programmer access to these child dialog controls. These are also grandchildren of the dialog.

A dialog control sends its event notifications to its direct parent, never to its grandparent. The *isGrandchild* method of the **`Edit`** control is a special case method. Its purpose is to allow the ooDialog framework to set up some special handling so that four event notifications will be intercepted and used to invoke a method in the Rexx dialog. Three of the methods are keydown events, the TAB, RETURN and ESCAPE keydown events. These events occur when the user types the return, or enter, key, the escape key, or the tab key. The fourth event is the KILLFOCUS event. The event occurs when the edit control loses the focus.

Details

Requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version*() method to determine the current version of the library. A syntax error is raised if this method is invoked on an operating system where it is not supported.

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example sets the parent of an edit control to be a list-view. It then invokes *isGrandchild* so that the *onEditGrandchildEvent* method will be invoked in the Rexx dialog for the KILLFOCUS, RETURN keydown, and ESCAPE keydown events:

```
list = self~newListView(IDC_LISTVIEW)
```

```
edit = self~newEdit(IDC_EDIT)
edit~setParent(list)
edit~isGrandchild
```

13.21. isModified

```
>>--isModified-----------------------------------><
```

Determines if the text in the edit control has been modified.

Arguments:

There are no arguments for this method.

Return value:

Returns `.true` if the edit control's modified flag is set, otherwise `.false` .

Remarks:

The operating system automatically sets the modification flag to false when the edit control is created. If the user changes the control's text, then the operating system sets the flag to true. The *setModified* method can be used to set the flag to true or false.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example saves the file in the edit control when the user ends the dialog by using the *Ok* button.

```
::method ok unguarded

if self~newEdit(IDC_EDIT_FILE)~isModified then self~saveFile
return self~ok:super
```

13.22. isSingleLine

```
>>--isSingleLine---------------------------------><
```

Determines if this edit control is a single line or a multi-line edit control.

Arguments:

There are no arguments to this method.

Return value:

Returns `.true` if this edit control is a single line edit control and `.false` if it is a multi-line edit control.

Remarks:

Some methods are not applicable for single line edit controls, or behave differently between single line and multi-line edit controls. For instance, the *setTabStops* method does nothing for a single line edit control. The *isSingleLine* method allows the programmer to write generic code for an edit control by checking which type of edit control the code is dealing with.

Example:

This example ...

```
::method setAppDefaultTabs private
  use strict arg editControl

  if editControl~isSingleLine then return .false

  editControl~setTabStops(.array~of(10, 20, 30, 40))
  return .true
```

13.23. lineFromIndex

```
>>--lineFromIndex(--+-------------+--)-----------><
                    +--charIndex--+
```

Gets the line index of the line that contains the character at the specified character index.

Arguments:

The single argument is:

charIndex [optional]

The non-negative, one-based, character index of the character whose line index is needed.

If *charIndex* is 0, then this method retrieves the current line index, or, if there is a selection, the line containing the beginning of the current selection. The current line is the line the caret is on. The default if *charIndex* is omitted is 0.

Return value:

The one-based line index containing the specified character. If the *charIndex* is beyond the end of the text in the edit control, the index of the last line is returned.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example puts some text into a multiline edit control and then displays the line index that contains the character at index 55.

```
msg = "It is easy to learn and easy to use." || .endOfLine || -
      "Have fun with it!"

edit = self~newEdit(200)
edit~setText(msg)

say "Character 55 is contained in line" edit~lineFromIndex(55)

/* Output would be:

Character 55 is contained in line 2

*/
```

13.24. lineIndex

```
>>--lineIndex(--+---------+--)-------------------><
                +--index--+
```

Determines the one-based character index of the first character in the line specified.

Arguments:

The single argument is:

line [optional]

The one-based index of the line whose character index needs to be determined. If *line* is 0 then the character index of the first character in the current line is returned.

Return value:

On success, the one-based character index for the first character in the line specified. 0 is returned on error. In particular, 0 is returned if *line* is greater than the number of lines in the edit control.

Remarks:

The character index is the one-based index of the character from the beginning of the edit control. This includes characters like line feed and carriage return that are not displayed on the screen. The current line is the line the caret is on.

This method is intended for multiline edit controls, but it works for single line edit controls, returning 1 when *line* is 1 or 0 and returning 0 for any other value of *line*

Example:

The following example sets the text for edit control with the *symbolic* id of IDC_EDIT and displays the number of text lines (3), the starting index of the second line (carriage return, 0x0d, and line feed, 0x0a, which mark a line break, are also considered to be characters), and the length of the third line:

```
editControl = self~newEdit(IDC_EDIT)

text = "It is easy to learn and easy to use." || '0d0a'x || -
       "Have fun with it!"                    || '0d0a'x || -
       "But don't over do it."

editControl~setText(text)

say "Number of lines:" editControl~lines
say "Line 2 begins at index" editControl~lineIndex(2)
say "Length of 3rd line:" editControl~lineLength(3)

/* Output would be:

Number of lines: 3
Line 2 begins at index 39
Length of 3rd line: 21
*/
```

13.25. lineLength

```
>>--lineLength(--+---------+--)------------------><
                 +--index--+
```

Returns the length, in characters, of the specified line in an edit control.

Arguments:

The only argument is:

index [optional]

The one-based index of the line whose length is needed. This argument can also be 0, see the remarks.

Return value:

On success, the number of characters in the line specified. On error -1. In particular, -1 is returned if *lineIndex* is greater than the number of lines in the edit control.

Remarks:

The length of the line does not include the carriage return and line-feed characters at the end of the line.

If *lineIndex* is 0 the return changes in this fashion:

If there is no selection, the number of characters in the current line is returned. The current line is the line with the caret in it.

If there is a selection, this method returns the number of unselected characters on lines containing selected characters.

For example, if the selection started at the fourth character on one line through the next line up through the fourth character from the end of the line, the return value would be 6. The first 3 characters on the first line and the last 3 on the next line.

Example:

This example demonstrates some uses of line length:

```
ec = self~newEdit(IDC_EDIT_INTRODUCTION)

text = "ooRexx is a great programming language." || .endOfLine || -
       "It is easy to learn and easy to use."    || .endOfLine || -
       "Have fun with it!"                       || .endOfLine || -
       "But don't over do it."

ec~setText(text)

say "Number of lines:" edit~lines

-- The user has the caret on the 3rd line with no selection:
say "Number of characters in current line:" ec~lineLength
say "Number of characters in line 4:" ec~lineLength(4)
say "Number of characters in line 5:" ec~lineLength(5)

-- Now say the user has started a selection on the i of 'is' on the second
-- line and extended the selection up through the space preceding the 'it'
-- ont the 3rd line:
say "Number of unselected characters in lines with a selection:" ec~lineLength

/* Output would be:

Number of lines: 4
Number of characters in current line: 17
Number of characters in line 4: 21
Number of characters in line 5: -1
Number of unselected characters in lines with a selection: 6
*/
```

13.26. lines

```
>>--lines----------------------------------------><
```

Determines the number of lines in a multiline edit control.

Arguments:

There are no arguments for this method.

Return value:

The return is the number of lines in the edit control. If there is no text in the control the return is 1. The return will never be less than 1.

Remarks:

The *lines* method returns the total number of text lines, not just the number of visible lines. If the dialog is resizable, the number of lines can change when the size of the edit control changes.

Example:

The *lineIndex* example uses the *lines* method

13.27. lineScroll

```
>>--lineScroll(--cChars--,--cLines--)------------><
```

Scrolls the text horizontally and vertically in a multiline edit control.

Arguments:

The arguments are:

cChars [required]

The number of characters to scroll horizontally. Positive numbers scroll the text to the left and negative numbers scroll the text to the right

cLines [required]

The number of lines to scroll vertically. Positive numbers scroll the text up and negative numbers scroll the text do down.

Return value:

The method always returns 0 for a multiline edit control. It returns 1 for a single-line edit control to indicate that this method does not apply to single-line edit controls.

Remarks:

The edit control does not scroll vertically past the last line of text. If the current line, plus the number of lines specified by *cLines*, exceed the total number of lines in the edit control, the last line of the edit control is scrolled to bottom of the edit control. The *lineScroll* method can, however, be used to scroll horizontally past the last character of a line, but it will not scroll past the width of the longest line in the edit control

Example:

This example scrolls an edit control by 50 characters and 35 lines:

```
edit~scroll(50, 35)
```

13.28. margins

```
>>--margins--------------------------------------><
```

Returns the left and right margins of the edit control as a **`String`** object.

Arguments:

This method has no arguments

Return value:

Returns the left and right margins, separated by a space. The values are in pixels.

Remarks:

The *margins* method is exactly equivalent to the *getMargins* method. The values returned will be exactly the same whichever method is used.

13.29. noContextMenu

```
>>--noContextMenu(--+--------+--)----------------><
                    +--undo--+
```

Removes the edit control's internal context menu. If the context menu was removed, it can be restored.

Arguments:

The only argument is:

undo [optional]

If *undo* is **`.true`** the edit control's context menu is restored. If **`.false`** the context menu is removed. The default is **`.false`**.

Return value:

`.true` on success, otherwise **`.false`**.

Remarks:

Edit controls have a built-in context menu that allows the user to move text between the edit control and the clipboard. The context menu appears when the user right-clicks the control. The commands in the context menu include Undo, Cut, Copy, Paste, Delete, and Select All.

However, if the Rexx programmer wants to use his own context menu through the *PopupMenu* class, the built-in menu will prevent that. Removing the built-in menu through the *noContextMenu* allows the programmer to replace the built-in menu with their own context menu. If the context menu is removed and the programmer does not supply a context menu, then no context menu is displayed when the user right-clicks on the edit control.

Details

Requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version*() method to determine the current version of the library. A syntax error is raised if this method is invoked on an operating system where it is not supported.

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*. The following codes may be set. They are set by the ooDialog framework rather than the operating system and are used to indicate error conditions:

ERROR_NOT_SUPPORTED (50)

The request is not supported. **Meaning:** The method was asked to remove the edit control's context menu when it was already removed, or asked to restore the edit control's context menu when it was not removed to begin with.

ERROR_SIGNAL_REFUSED (156)

The recipient process has refused the signal. **Meaning:** The method tried to remove, or restore, the context menu, but failed. This is unlikely.

13.30. pasteText

```
>>--pasteText------------------------------------><
```

Copies the current content of the clipboard to the edit control at the current caret position.

Arguments:

There are no arguments for this method.

Return value:

This method always returns 0.

Remarks:

The paste operation only occurs if the current content of the clipboard is in text format.

13.31. removeStyle

```
>>--removeStyle(--style--)-----------------------><
```

Removes one or more edit control styles from the edit control.

Arguments:

The single argument is:

style [required]

A list of 1 or more of the following keywords separated by spaces, case is not significant:

UPPER	WANTRETURN	GROUP
LOWER	OEM	
NUMBER	TAB	

The *argument* list for the *addStyle* method details the meaning of the keywords. The *createEdit* method documents the meaning of all the edit control style keywords.

Return value:

The positive numeric value of the edit control's previous style.

It is possible, but very unlikely, that a negative number is returned. See the remarks.

Remarks:

This method only works with the window *styles* that can be changed after the edit control has been created. Not all window styles can be changed after the control is created.

If an operating system error occurs, this method returns the negated value of the system error code. However, this method also sets **`.systemErrorCode`** if there is an error.

The negative number return is a hold over from earlier versions of ooDialog before **`.systemErrorCode`** was introduced.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example shows the syntax to remove the lower and the tabstop styles from an edit control.

```
ec = dlg~newEdit(IDC_EDIT_SHIPPING_CODES)
ec~removeStyle("LOWER GROUP")
```

13.32. replaceSelText

```
>>--replaceSelText(--text--+-----------+--)------><
                           +-,-canUndo-+
```

Replaces the text selection in an edit control with the specified text.

Arguments:

The arguments are:

text [required]

The text used for replacement.

canUndo [optional]

If **`.true`** the replacement can be undone. If **`.false`**, the replacement can not be undone. The default is **`.true`**.

Return value:

This method always returns 0.

Remarks:

Use the *replaceSelText* to replace only a portion of the text in an edit control. To replace all of the text, use the *setText* method.

If there is no selection, the replacement text is inserted at the cursor.

Example:

This example ...

```
  edit = self~newEdit(IDC_EDIT_INTERPRETER)

  edit~setText("Object Rexx is a hybrid language.")
  edit~select(17, 25)

  edit~replaceSelText("n interpreted")
  say edit~getText

/* output would be:

  Object Rexx is an interpreted language.
```

```
*/
```

13.33. replaceStyle

```
>>--replaceStyle(--removedStyle--,--addedStyle--)--------------><
```

Removes and adds one or more edit control styles in one operation.

Arguments:

The arguments are:

removedStyle [required]

One or more of the edit control style keywords. The allowable keywords are the same for *removedStyle* and *addedStyle*.

addedStyle [required]

One or more of the edit control style keywords. The allowable keywords are the same for *removedStyle* and *addedStyle*.

Style keywords

A list of 1 or more of the following keywords separated by spaces, case is not significant:

UPPER	WANTRETURN	GROUP
LOWER	OEM	
NUMBER	TAB	

The *argument* list for the *addStyle* method details the meaning of the keywords. The *createEdit* method documents the meaning of all the edit control style keywords.

Return value:

The positive numeric value of the edit control's previous style.

It is possible, but very unlikely, that a negative number is returned. See the remarks.

Remarks:

This method only works with the window *styles* that can be changed after the edit control has been created. Not all window styles can be changed after the control is created.

If an operating system error occurs, this method returns the negated value of the system error code. However, this method also sets **`.systemErrorCode`** if there is an error.

The negative number return is a hold over from earlier versions of ooDialog before **`.systemErrorCode`** was introduced.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

The following would change an edit control from one that only allows numbers to one that uppercases all letters entered, while removing the tabstop style and adding the group style.

```
  ed = dlg~newEdit(IDC_EDIT_MIXED)
```

```
ec~replaceStyle("NUMBER TAB", "UPPER GROUP")
```

13.34. scrollCommand

```
>>--scrollCommand(--+-------+--+----------------+--)----------><
                    +--cmd--+  +-,-repetitions--+
```

Scrolls the text vertically in a multiline edit control.

Arguments:

The arguments are:

cmd [optional]

Exactly one of the following keywords, case is not significant. The default if this argument is omitted is UP. If an unrecognized keyword is used, then UP is substituted for the keyword.

UP	DOWN	PAGEUP	PAGEDOWN

UP

The edit control is scrolled up one line.

DOWN

The edit control is scrolled down one line.

PAGEUP

The edit control is scrolled up one page.

PAGEDOWN

The edit control is scrolled down one page.

repetitions [optional]

The number of times to perform the scroll command. The default is 1.

Return value:

The return is the value returned by the operating system.

The operating system packs the success indicator (**`.true`** or **`.false`**) in the high-word of the return. The low-word contains the number of lines scrolled. Use the *sHiWord* and *sLoWord* methods to extract these values

Remarks:

For the UP and PAGEUP commands, the number of lines scrolled by the operating system is negative. Be sure to use the *signed* version of the loWord method, *sLoWord*, to get the correct value.

When the *repetitions* argument is used, ooDialog issues the scroll command in a loop for the number of *repetitions* specified. The returned value is then the return from issuing the last command. Because of this, the return when the *repetitions* argument is used is not meaningful. For instance, when using this invocation: **`ret = editControl~scrollCommand("PAGEUP", 15)`** when the edit control only has 4 pages of text will indicate that the command failed and that 0 lines were scrolled.

The method has no effect on single-line edit controls.

Please **note**, older versions of the ooDialog documentation also listed the LEFT, RIGHT, PAGELEFT, and PAGERIGHT commands for this method. These command keywords are misleading, the text can only be scrolled vertically with this method. LEFT is the same as UP, RIGHT is the same as DOWN, etc..

Example:

This example shows a method that scrolls the text in the edit control all the way to the top. The method is connected to a *Home* menu item in the application.

```
::method onHomeMenuItem
  expose editControl

  success = .true
  do while success
    ret = editControl~scrollCommand("PAGEUP")
    success = .DlgUtil~sHiWord(ret)
  end
```

13.35. select

```
>>--select(--+---------+--+--------+--)----------><
             +--start--+  +-,-end--+
```

The *select* method selects text, and sets the position of the cursor, within the edit control. See the remarks section for more detail on how the *start* and *end* values effect the behavior.

Arguments:

The arguments are:

start [optional]

Specifies a one-based character position in the selection, normally the starting position of the selection. See the remarks section because the actual behavior is dependent on the value of both *start* and *end*. If omitted the default is 1.

end [optional]

Specifies a one-based character position in the selection, normally the ending position of the selection. See the remarks section because the actual behavior is dependent on the value of both *start* and *end*. If omitted the default is 0.

Return value:

This method always returns 0.

Remarks:

The *start* value can be greater than the *end* value. The lower of the two values specifies the character position of the first character in the selection. The higher value specifies the position of the first character beyond the selection. The cursor is placed at the position of the first character beyond the selection, provided one of the other special cases does not apply.

If the starting position equals the ending position, no text is selected and the cursor position is set to the character specified by the *start* argument. If the ending index is 0 and the starting index is 1, the entire text is selected and the cursor is placed at the position immediately following the end of the text.

If *start* is 0, the *end* argument is ignored, any selection is removed, and the cursor is not moved.

Notice that if both arguments are omitted, the default values for *start* and *end* will result in all of the text being selected.

Example:

See the *selected* method's example.

13.36. selected

```
>>--selected-------------------------------------><
```

Returns a **`String`** object containing the starting and ending position of the text selection in the edit control.

Arguments:

This method has no arguments.

Return value:

The one-based character position of the start and end of the current selection, separated by a blank.

Remarks:

If the starting position equals the ending position, no text is selected and the position specifies the current cursor position.

Note that the ending position specifies the position of the first character *beyond* the selection, not the last character position in the selection. All the text is selected if the starting position is 1 and the ending position is 1 greater than the length of the text in the edit control.

Example:

A simple example:

```
edit = self~newEdit(IDC_EDIT)

parse value edit~selected with start end
if start == end then do
  say "There is no text selected"
end
else do
  say "Starting character position of the selection is" start
  say "Last character position of the selection is" end - 1
end

edit~select(1, 0)  -- Selects all the text.
```

13.37. selection

```
>>--selection------------------------------------><
```

Returns a **`Directory`** object containing the starting and ending position of the text selection in the edit control.

Arguments:

The are no arguments for this method.

Return value:

A **`Directory`** object with the indexes *startChar* and *endChar*. The *startChar* index is the one-based character position of the first character in the selection. The *endChar* index is the one-based character position of the first position past the end of the selection.

Remarks:

If the starting position equals the ending position, no text is selected and the position specifies the current cursor position.

Note that the ending position specifies the position of the first character *beyond* the selection, not the last character position in the selection. All the text is selected if the starting position is 1 and the ending position is 1 greater than the length of the text in the edit control.

Example:

A simple example:

```
s = self~newEdit(IDC_EDIT)~selection

if s~startChar == s~endChar then do
  say "There is no text selected"
end
else do
  say "Starting character position of the selection is" s~startChar
  say "Last character position of the selection is" s~endChar - 1
end
```

13.38. setCue

```
>>--setCue(--text--+---------+--)----------------><
                   +-,-show--+
```

Sets the text for *cue* banner text.

Arguments:

The arguments are:

text [required]

The text for the cue. This string must be 255 characters or less in length.

show [optional]

Whether the text show disappear when the edit control gains the focus, or whether it should still show. The default is **`.false`** the text disappears when the edit control gains the focus. If *show* is **`.true`**, on Windows Vista or later the cue text will remain until the user enters some characters into the edit control.

Return value:

Zero on success, one on failure.

Remarks:

Cue text provides a visual hint to the user as to the purpose of the edit control. For example the text of an edit control that is used to begin a search might have the cue text: *Enter search here*. The cue is displayed in gray text. Once some text is entered into the edit box, the cue no longer displays.

Normally when the user clicks the on the edit control, the cue text goes away. However on Windows Vista and later this behavior can be changed so that the cue text remains until the user actually types some characters in the edit control.

It is not possible to set a cue on a multiline edit control.

Details

Raises syntax errors when incorrect arguments are detected.

Requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version*() method to determine the current version of the library. A syntax error is raised if this method is invoked on an operating system where it is not supported.

13.39. setLimit

```
>>--setLimit(--count--)--------------------------><
```

Sets the maximum number of characters the user can type in the edit control.

Arguments:

The single argument is:

count required

A non-negative number that is the limit for the number of characters a user can type into the edit control. See the remarks for the meaning of 0.

Return value:

This method always returns 0.

Remarks:

Note that the text limit is the number of characters the user can type in the edit control. The limit does not effect any text already entered in the edit control. It also does not effect the length of the text that the *setText* method can copy into the edit control. If a program uses *setText* to copy more text into the edit control than the text limit, the user can edit the entire contents of the edit control.

If *count* is 0, then the limit for single line edit controls is set to 2,147,483,647 characters and for multiline edit controls it is set to 4,294,967,295 characters.

Before the *setLimit* method is used, the default for the text limit of an edit control is 32,767 characters.

13.40. setMargins

```
>>--setMargins(--+--------+--+----------+--)-----><
                 +--left--+  +-,-right--+
```

Sets the left and right margins of the edit control as specified.

Arguments:

The arguments are:

left [optional]

Specifies the left margin in pixels. If this argument is omitted, the left margin is not changed. If both the *left* and *right* arguments are omitted, the margins are set to a narrow margin calculated from the font info of the edit control.

right [optional]

Specifies the right margin in pixels. If this argument is omitted, the right margin is not changed. If both the *left* and *right* arguments are omitted, the margins are set to a narrow margin calculated from the font info of the edit control.

Return value:

This method always returns 0.

Remarks:

The *setMargins* method will cause the edit control to redraw itself to reflect the new margins. If both arguments are omitted, the method causes the edit control to calculate a narrow left and right margin based on the font of the edit control. If no font is set for the edit control, this margin is set to 0.

Example:

This example ...

```
  editControl = self~newEdit(IDC_EDIT_LICENSE)

  editControl~setMargins(10, 5)
  m = editControl~getMargins
  say "The new left margin is" m~left "and the new right margin is" m~right

/* Output would be:

The new left margin is 10 and the new right margin is 5

*/
```

13.41. setModified

```
>>--setModified(--+---------+--)-----------------><
                  +--state--+
```

Sets the modified flag of an edit control to the state specified, **`.true`** or **`.false`** .

Arguments:

The single arguments is:

state [optional]

If *state* is **`.true`** the modified flag is set to true. If *state* is **`.false`** the flag is set to false. The default is **`.true`**

Return value:

This method always returns 0.

Remarks:

The operating system automatically sets the modification flag to false when the edit control is created. If the user changes the control's text, then the operating system sets the flag to true. The *setModified* method can be used to determine the current state of the flag.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example ...

```
::method saveLicense

    /* Write the license to a file for the user */
    self~writeLicense

    e = self~newEdit(IDC_EDIT_LICENSE)~setModified(.false)
```

13.42. setReadOnly

```
>>--setReadOnly(--+------+--)--------------------><
                  +-mode-+
```

Adds or removes the read only style for the edit control.

Arguments:

The single argument is:

mode [optional]

`.true` adds the read only style and `.false` removes the read only style. If the argument is omitted, the read only style is added.

Return value:

Returns 0 on success, and 1 if the method fails.

Remarks:

When an edit control has the read only style, the user can not modify the text in the edit control.

Details

Raises syntax errors when incorrect arguments are detected.

13.43. setRect

```
>>--setRect(--rect--+-----------+--)-------------><
                    +-,-redraw--+
```

Sets the formatting rectangle for the edit control, or resets it to the default.

Arguments:

The arguments are:

rect [required.]

A *Rect* object that contains the new coordinates for the formatting rectangle, or 0. If *rect* is 0, the formatting rectangle is reset to its default.

redraw [optional]

True or false, indicating if the edit control should redraw itself or not. The default is true. When *redraw* is true, the edit control immediately redraws itself after changing the formatting rectangle. When false, the redrawing is not done.

Return value:

This method always returns 0.

Remarks:

The visibility of the text in an edit control is dependent on its window rectangle and its formatting rectangle. The window rectangle is the client area of the window containing the edit control. The formatting rectangle is an artificial construct maintained by the operating system and used to format the text. The formatting rectangle can be set to be larger than the window rectangle, which will limit what can be seen of the text. Or the formatting rectangle can be made smaller than the window rectangle, which will create more white space around the text.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example gets the current formatting rectangle, then decreases its size by 4 pixels. The edit control is signaled to redraw itself.

```
::method addMoreBorder private
  use strict arg editCtrl

  fRect = editCtrl~getRect
  fRect~left    -= 4
  fRect~top     -= 4
  fRect~right   -= 4
  fRect~bottom -= 4

  -- Second argument is not needed, .true is the default.
  editCtrl~setRect(fRect, .true)
```

13.44. setTabStops

```
>>--setTabStops(--tabStops--)--------------------><
```

Sets the tab stops for text copied into a multi-line edit control.

When text is copied to the control, any tab character in the text causes space to be generated up to the next tab stop. This method is ignored if the edit control is a single-line edit control.

Arguments:

The single argument is:

tabStops

An array containing the tab stops. Each tab stop is a positive number expressed in dialog template units. If the array contains no elements, default tab stops are set at every 32 dialog template units.

If the array contains only 1 element at index 1, tab stops are set at every n dialog template units, where n is the distance at index 1.

Otherwise, tab stops are set to the numbers contained in the array.

Return value:

True if all tab stops were set, otherwise false.

Remarks:

The array must contain all positive numbers (or be an empty array to set the default tab stops.) Also, the array must not be sparse, i.e., it must not skip any array indexes.

The operating system will not do negative tab stops. I.e. you can not do 15 35 20. Also, note that when specifying an array of more than 1 tap stop, each tab stop is the absolute position of the tab stop, not the distance between the tab stops.

Under normal circumstances there is no way for a user to enter a tab character by typing in a multi-line edit control. The edit control would need to be sub-classed and this is not provided by ooDialog. However, the user could use copy and paste to paste in text with tabs. And, text placed in the edit control by the programmer, (for example by using the *setText* method,) can also contain tabs.

Details

Raises syntax errors when incorrect arguments are detected.

13.45. showBalloon

```
>>--showBalloon(--title,--text--+----------+--)--><
                                +-,--icon--+
```

Displays a balloon tip information window for the edit control.

Arguments:

The arguments are:

title [required]

A title for the balloon window. Titles are limited to 99 characters total in length.

text [required]

The text of the balloon window. This text is limited to 1023 characters in length. The programmer can use line break characters to control the formatting of the text.

icon [optional]

Used to specify which icon is displayed on the title line. The default is the informational icon. The following keywords can be used to specify another icon, or no icon. Only the first letter of the keyword is needed and case is insignificant:

ERROR

The error icon is used.

WARN

The warning icon is used.

NONE

No icon is displayed.

Return value:

Returns 0 on success, 1 if the method fails.

Remarks

Balloon tip windows are similar in appearance to the balloon captions used in comic books to display the dialog of the characters. They are also similar, but not the same, as tool tips. For an edit control the balloon tip looks like it is attached to the control.

The balloon is displayed for 10 seconds and then goes away. The programmer can also dismiss the balloon sooner using the *hideBalloon* method. Typically balloon windows are used to convey information to the user without having to put up a message box that the user then needs to dismiss manually.

Details

Raises syntax errors for incorrect usage.

Requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version*() method to determine the current version of the library. A syntax error is raised if this method is invoked on an operating system where it is not supported.

Example:

The following code snippet could come from a fictitious inventory program. An edit control is used to enter a product ID. If the user clicks on the edit control a ballon tip is shown displaying extended product information. (In a real application, the extended information might be based on what the user had entered in the edit control.)

Text is set in the edit control to give the user a 'cue' that she can click on the edit control for extra information. When the ballon tip is shown, it will automatically be dismissed after 10 seconds. This application connects a notification for the lost focus event. When the edit control loses focus, the ballon tip is hidden by the program.

```
  ...

::method defineDialog

  ...

  self~createStaticText(200, 10, 125, 150, 10, , "Enter product ID:")
  self~createEdit(IDC_ENTRYLINE, 10, 135, 150, 10, "AUTOSCROLLH", "cEntry")

  ...

::method initDialog
  expose editControl hwndEditControl

  editControl = self~newEdit(IDC_ENTRYLINE)
  hwndEditControl = editControl~hwnd

  self~connectEditEvent(IDC_ENTRYLINE, GOTFOCUS, "onFocus")
  self~connectEditEvent(IDC_ENTRYLINE, LOSTFOCUS, "onLostFocus")

  editControl~setCue("Click here for extended product information")

  -- Capture the mouse activate messages: WM_MOUSEACTIVATE message == 0x0021
  self~addUserMsg(onMouseActivate, "0x0021", "0xFFFFFFFF", 0, 0, 0, 0)

  ...

::method onMouseActivate
  expose editControl hwndEditControl

  if self~getFocus == hwndEditControl then do
    title = "Extended Product Information"
    text = "Product ID:"   || "9"x    || "4538-D32"       || .endOfLine || -
           "In stock:"     || "9"x    || "789"            || .endOfLine || -
           "Back ordered:" || "9"x    || "No"             || .endOfLine || -
           "EOL:"          || "0909"x || "January 2008"   || .endOfLine || -
           "Client:"       || "0909"x || "AT&T"           || .endOfLine || .endOfLine -
           "Notes: This product has served a limited number of customers" -
           "and should be considered archaic.  It will be replaced by a"   -
```

```
            "superior product line."

    editControl~showBalloon(title, text, "e");
  end

::method onLostFocus
  expose editControl

  -- When the edit control loses the focus, dismiss the balloon.
  editControl~hideBalloon
```

13.46. tab=

```
>>--tab = tabStops-------------------------------><
```

The *tab* = method is an alias for the *setTabStops* method. It behaves exactly the same except there is no return value.

13.47. undo

```
>>--undo-----------------------------------------><
```

Undoes the last edit control operation in the edit control's undo queue.

Arguments:

There are no arguments to the method.

Return value:

The return is always **`.true`** from a single-line edit control. For a multiline edit control, the return is **`.true`** if the operation succeeded, otherwise **`.false`**.

Remarks:

As long as there is no intervening edit operation, an undo operation can also be undone. For example you could restore some deleted text with the first invocation of the *undo* method, and then remove the text again with a second invocation of the *undo* method.

Details

Raises syntax errors when incorrect arguments are detected.

13.48. wantReturn

```
>>--wantReturn(--+--------------+--)-------------><
                 +--methodName--+
```

Connects the *Enter* keypress to a method in the Rexx dialog and prevents the edit control from sending the keypress to the dialog manager, which closes the dialog.

Arguments:

The single argument is:

methodName [optional]
: The name of the method in the Rexx dialog that will be invoked when the user press the enter key in the single-line edit control. If this argument is omitted, the method name is set to *onReturn*.

Return value:

Returns true on success, false on error.

Remarks:

By default in Windows dialogs, when the user presses the enter key, the default button is pressed. Normally the default button is the Ok button and the dialog is closed. Usually, it is not a good idea to try and change this behavior. Most Windows users become to depend on the behavior.

However, in rare cases, the programmer may want to change this behavior. Microsoft itself, sometimes changes this behavior, as can be seen when in-place editing of list-view labels is done. When the user is editing a list-view label in-place, hitting the enter key does not close the dialog, but instead accepts the current text as what the user wants. The *wantReturn* method can be used in those circumstances.

A method is always connected to the Enter keypress, and invoking the *wantReturn* method always prevents the Enter keypress from pressing the default button. If the programmer does not need to do anything in the event handling method, she can simply return 0 immediately from the method.

This method is only for single-line edit controls. If the edit control is a multi-line edit control, the *wantReturn* method fails and returns false.

Example:

This example invokes the *wantReturn* method. Note that the optional *methodName* argument is not used. Therefore the *onReturn* method will be connected:

```
::method setUpItemEdit private
  expose list itemEdit

  itemEdit~wantReturn
  ...
```

13.48.1. wantReturn Event Handler

The event handler for the want return event is invoked when the user presses the Enter, sometimes called the Return, key when the connected, single-line, edit control has the focus.

The programmer must return a value from the event handler and the interpreter waits for this return.

```
::method onWantReturn unguarded
  use arg ctrlID, editControlObj

  return 0
```

Arguments:

The event handling method receives 2 arguments:

ctrlID
: The resource *ID* of the edit control.

editControlObj
: The Rexx Edit control object.

Return:

The event handler must return a value, but the operating system ignores the value of the return, so any value is okay. Returning 0 is a good choice.

Example

The following example shows an event handler for the return event. The dialog has a single-line entry field that lets the user type in record number and advance to that record. Hitting enter signals that the user wants to advance to record entered in the edit control.

Note that the processing of retrieving the new record could, theoretically, be lengthy. So, the event handler replies early to prevent the dialog from appearing hung while the processing going on:

```
::method onReturn unguarded
  expose list currentItem dbMgr
  use arg id, editObj

  itemIndex = editObj~getText
  max = list~items

  reply 0

  msg = ''
  if itemIndex < 1 then msg = 'List-view record item can not be less than 1'
  else if itemIndex > max then msg = 'List-view record item can not be greater than' max

  if msg \== '' then do
    editObj~setText(currentItem)
    editObj~showBalloon('Entry Error', msg, 'ERROR')
    return
  end

  oldItem = currentItem
  currentItem = itemIndex - 1

  lvRow = list~getItemData(currentItem)
  rec = dbMgr~getRecordForID(lvRow~userData)

  if rec == .nil then do
    say 'Failed to get new record' -- TODO need message box
    currentItem = oldItem
    editObj~setText(currentItem)
    return
  end

  self~setRecordFields(rec)
  self~setNavigateButtons

  self~setStateToUnchanged
```

List Box Controls

The **`ListBox`** class provides methods to query and modify list box controls.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with list box controls:

Instantiation:

Use the *newListBox*() method of the *dialog* object to retrieve a list box object.

Dynamic Definition:

To dynamically create a list box in the dialog template of a *UserDialog* use the *createListBox* method.

Event Notification

To connect the *event* notifications sent by the underlying list box control to a method in the Rexx dialog object use the *connectListBoxEvent*) method.

14.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with ListBox objects, including the pertinent methods from other classes:

Table 14.1. ListBox Methods and Attributes

Method	Documentation
Useful External Methods	
newListBox	Obtains a ListBox object that represents a list box control in a dialog.
createListBox	Creates a list box control in an *UserDialog*.
connectListBoxEvent	Connects ListBox event notifications to a Rexx dialog method.
Instance Methods	
add	Adds a new item to the list.
addDirectory	Adds all or selected file names of a given directory to the list box.
columnWidth=	Sets the width, in dialog units, of the list box columns in a multicolumn list box control. **(Inaccurate.)**
columnWidthPx=	Sets the width, in pixels, of the list box columns in a multicolumn list box control.
delete	Removes a list item from the associated list box.
deleteAll	Removes all list items from the associated list box.
deSelectIndex	Deselects the list entry at the specified position for multiple selection listboxes.
deSelectRange	Removes the selection for one or more consecutive items of the associated multiselection list box.
find	Searches the list box for a list entry containing a specified text.
getFirstVisible	Retrieves the index of the first visible list item in the list box.
getText	Gets the text of the list item at the specified position in the list box.
hitTestInfo	Determines the one-based index of the item nearest the specified point in this list box.

Method	Documentation
insert	Inserts a new item into the list after the specified item.
itemHeight	Retrieves the height, in dialog units, of the items in the list box. **(Inaccurate.)**
itemHeightPx	Retrieves the height, in pixels, of the items in the list box.
itemHeight=	Sets the height, in dialog units, of the items in the list box. **(Inaccurate.)**
itemHeightPx=	Sets the height, in pixels, of the items in the list box.
items	Retrieves the number of items in the list box.
makeFirstVisible	Scrolls the list box so the specified item is at the top of the visible range.
modify	Changes the text of the list item at the specified position.
select	Selects the list entry that matches the specified text.
selected	Retrieves the text of the currently selected list entry.
isSingleSelection	Determine if a list box is a single-selection list box.
selectedIndex	Retrieves the index of the currently selected list entry.
selectedIndexes	Retrieves the indexes of all items that are currently selected in the associated multiselection list box.
selectedItems	Retrieves the number of items that are currently selected in the associated multiselection list box.
selectIndex	Selects the list entry at the specified position.
selectRange	Selects one or more consecutive items of the associated multiselection list box.
setTabulators	Sets the tabulators for the associated list box control.
setWidth	Sets the internal width, in dialog units, of the list box. **(Inaccurate.)**
setWidthPx	Sets the internal width, in pixel, of the list box.
width	Retrieves the internal width, in dialog units, of the associated list box. **(Inaccurate.)**
widthPx	Retrieves the internal width, in pixels, of the associated list box.

14.2. newListBox (dialog object method)

`ListBox` objects can not be instantiated by the programmer from Rexx code using a *new*() method. Rather, a list box object is obtained by using the *newListBox*() method of the *dialog* object. The syntax is:

```
>>--newListBox(--id--)---------------------------><
```

14.3. createListBox (UserDialog method)

A list box control can be created in the dialog template of a *UserDialog* through the *createListBox* method. The basic syntax is:

```
>>--createListBox(-id-,--x-,--y-,--cx-,--cy-+----------+-+-------------+--)----><
                                            +-,--style-+ +-,--attrName-+
```

14.4. connectListBoxEvent (dialog object method)

To connect event notifications from a list box control use the *connectListBoxEvent* method of the *dialog* object. The basic syntax is:

```
>>--connectListBoxEvent(--id--,--event--+---------------+--)------------------><
                                        +--,-methodName--+
```

14.5. add

```
>>--add(--listEntry--)---------------------------><
```

The add method adds a new item to the list. If the list is not sorted, the new item is added to the end of the list.

Arguments:

The only argument is:

listEntry

A text string added to the list.

Return value:

A one-based index that specifies the position at which the entry has been added, or 0 or a value less than 0 to indicate an error.

14.6. addDirectory

```
>>--addDirectory(--drvPath--+-------------------+--)--------------------------><
                            +-,-fileAttributes--+
```

The addDirectory method adds all or selected file names of a given directory to the list box.

Arguments:

The arguments are:

drvPath [required]

The drive, path, and name pattern.

fileAttributes [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant:

READWRITE	HIDDEN	DIRECTORY
READONLY	SYSTEM	ARCHIVE

READWRITE

Normal read/write files. This is the default if this argument is omitted.

READONLY

Files that have the read-only bit.

HIDDEN

Files that have the hidden bit.

SYSTEM
: Files that have the system bit.

DIRECTORY
: Files that have the directory bit.

ARCHIVE
: Files that have the archive bit.

Return value:

The one-based index of the file added last to the list, or 0 if an error occurred.

Example:

The following example puts the names of all read/write files with extension .REX in the given directory of the list box:

```
self~addDirectory(203, drive":\"path"\*.rex", "READWRITE")
```

14.7. columnWidth=

```
>>--columnWidth = width--------------------------><
```

The columnWidth= method sets the width of the list box columns in a multicolumn list box control, in dialog units.

Arguments:

The only argument is:

width
: The width of all list box columns, in dialog units.

14.8. columnWidthPx=

```
>>--columnWidthPx = width------------------------><
```

The columnWidthPx= method sets the width of the list box columns in a multicolumn list box control, in pixels.

Arguments:

The only argument is:

width
: The width of all list box columns, in pixels.

14.9. delete

```
>>--delete(--+-------+--)------------------------><
             +-index-+
```

The delete method removes a list item from the associated list box.

Arguments:

The only argument is:

index

The index (starting with 1) of the list item to be removed from the list. If this argument is omitted, the currently selected item is deleted.

Return value:

The number of remaining list items, or 0 to indicate an error.

14.10. deleteAll

```
>>--deleteAll------------------------------------><
```

The deleteAll method removes all list items from the associated list box.

14.11. deSelectIndex

```
>>--deSelectIndex(--+-------+--)-----------------><
                    +-index-+
```

The deSelectIndex method deselects the list entry at the specified position for multiple selection listboxes. The currently selected list item is highlighted and surrounded by a dotted border.

Arguments:

The only argument is:

index

The index (starting with 1) of the list item to be deselected. If you specify no index or 0 for this argument, all items are deselected.

Return value:

-1 if an error occurred.

14.12. deSelectRange

```
>>--deSelectRange(--+------------+--+------------+--)------------------------><
                    +-startIndex--+  +-,--endIndex-+
```

The deSelectRange method removes the selection for one or more consecutive items of the associated multiselection list box.

Arguments:

The arguments are:

startIndex

The position of the first list item to be deselected. If this argument is omitted, the deselection range starts with the first item in the list.

endIndex
: The position of the last list item to be deselected. If this argument is omitted, the deselection range ends with the last item in the list.

If *startIndex* is equal to *endIndex*, a single item of the list is deselected.

Return value:

-1 if an error occurred. This can happen, for example, if the list box is no multiselection list box.

Example:

The following example deselects the items 1 through 5:

```
lb = self~newListBox("Offers")
if lb == .Nil then return
lb~deSelectRange(,6)
```

14.13. find

```
>>--find(--TextorPrefix--+--------------------------------+--)----------------><
                         +-,--+-----------+--+---------+-+
                              +-startIndex-+  +-,--exact-+
```

The find method searches the list box for a list entry containing the specified text or prefix. The search is caseless.

Arguments:

The arguments are:

TextorPrefix
: The text or prefix for which the list is searched.

startIndex
: The first list item at which the search is to be started. When the search reaches the bottom of the list, it is continued backward. If you omit this argument or specify 0, the entire list is searched.

exact
: If you specify 1 or E, the text of the list item must exactly match the text specified for *TextorPrefix*. If you omit this argument or specify 0, the list entries are searched for a prefix that matches *TextorPrefix*.

Return value:

The one-based index of the list entry that matches the search text, or 0 if not found.

14.14. getFirstVisible

```
>>--getFirstVisible------------------------------><
```

The getFirstVisible method retrieves the index of the first list item visible in the list box.

Return value:

The one-based index of the list box item visible first.

14.15. getText

```
>>--getText(--index--)---------------------------><
```

The getText method gets the text of the list item at the specified position in the list box.

Arguments:

The only argument is:

index

The one-based index of the list box item containing the text you are interested in.

Return value:

The text of the list box item at the given position, or an empty string if the *index* does not refer to an item or an error occurred.

14.16. hitTestInfo

```
Form 1:

>>--hitTestInfo(--pos--+---------+--)------------><
                       +-,-info--+

Form 2:

>>--hitTestInfo(--x--,--y--+---------+--)--------><
                           +-,-info--+

Generic Form:

>>--hitTestInfo(--hitPoint--+---------+--)-------><
                            +-,-info--+
```

Determines the one-based index of the item nearest the specified point in this list box.

Arguments:

The arguments are:

hitPoint [required]

The point, in client *coordinates*, to test.

info [optional in / out]

A **`.Directory`** object in which additional information concerning the hit test will be returned. This argument can be omitted if the additional information is not needed. When the argument is used, on return *info* will contain the following indexes:

INCLIENTAREA

The value of this index will be true or false. It will be true when the hit point is within the client area of the list box and false if the point is not within the client area.

ITEMINDEX

The index of the item closest to the point specified. This value will be the same as the return value from the method.

Return value:

The one-base index of the list box item that is closet to the point specified.

Remarks:

It is important to realize that no matter where the point specified is, there will *always* be an item that is closest to it. The return value of this method will always be the index of an item in the list box. Because of this, the *info* argument is usually needed. The *INCLIENTAREA* index is really the only way to know if a list box item is under the specified point.

Another point that may not be evident at first glance. The item index returned will always be the index of an item showing. That is, say the list box contains 12 items and the items are scrolled so that the items 1 through 3 are scrolled to the top of the list box out of view. If the specified point has a y coordinate of -5, it might be expected that the index returned will be 3, the item that would actually be under that point. The return will not be 3, it will be 4, the index of the item at the top of the viewable area in the list box.

The operating system accepts any point, even one that seems silly like (-7000, -6000).

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example returns the text of the list box item under the specified point, or .nil if the point is not over the list box:

```
::method textAtPoint
  use strict arg point

  lb = self~newListBox(IDC_LB_FNAME)

  d = .Directory~new
  index = lb~hitTestInfo(point, d)
  if d~inClientArea then do
    text = lb~getText(index)
    return text
  end

  return .nil
```

14.17. insert

```
>>--insert(--+-------+--,--listEntry--)----------><
             +-index-+
```

The insert method inserts a new item into the list after the specified item.

Arguments:

The arguments are:

index

The index (starting with 1) of the list item after which the new item is to be added. If this argument is omitted, the list entry is added after the currently selected item.

listEntry

A text string added to the list.

Return value:

A one-based index that specifies the position at which the entry has been added, or 0 or a value less than 0 to indicate an error.

14.18. isSingleSelection

```
>>--isSingleSelection----------------------------><
```

Determine if a list box is a single-selection list box. In a single-selection list box, the user can select only one item at a time.

Arguments:

This method has no arguments.

Return value:

Returns **`.true`** if the list box is a single-selection list box, otherwise **`.false`**.

14.19. itemHeight

```
>>--itemHeight-----------------------------------><
```

The itemHeight method retrieves the height of the items in the list box, in dialog units.

Return value:

The height of the list box items, in dialog units.

14.20. itemHeight=

```
>>--itemHeight = height--------------------------><
```

The itemHeight= method sets the height of the items in the list box, in dialog units.

Arguments:

The only argument is:

height

The height of all list box items, in dialog units.

14.21. itemHeightPx

```
>>--itemHeightPx---------------------------------><
```

The itemHeightPx method retrieves the height of the items in the list box, in pixels.

Return value:

The height of the list box items, in pixels.

14.22. itemHeightPx=

```
>>--itemHeightPx = height------------------------><
```

The itemHeightPx= method sets the height of the items in the list box, in pixels.

Arguments:

The only argument is:

height

The height of all list box items, in pixels.

14.23. items

```
>>--items----------------------------------------><
```

The items method retrieves the number of items in the list box.

Return value:

The number of items in the list box.

14.24. makeFirstVisible

```
>>--makeFirstVisible(--index--)------------------><
```

The makeFirstVisible method makes the list entry at the specified position the first visible list item when you scroll up or down.

Arguments:

The only argument is:

index

The one-based index of the list box item to be made first visible.

Return value:

-1 if an error occurred.

14.25. modify

```
>>--modify(--+-------+--,--newText--)------------><
             +-index-+
```

The modify method changes the text of the list item at the specified position in the list box.

Arguments:

The arguments are:

index

The one-based index of the list box item of which the text is to be changed. If you omit this argument, the currently selected item is modified.

newText

The new text string to be displayed at the given position.

Return value:

The one-based index of the modified list box item at the given position. The return value is 0 if an error occurred, or -1 if the *index* does not refer to an item.

14.26. select

```
>>--select(--itemText--)-------------------------><
```

The select method selects the list entry that matches the specified text.

Arguments:

The only argument is:

itemText

The text that the list box is searched for.

Return value:

0 if an error occurred or a matching list entry was not found.

14.27. selected

```
>>--selected-------------------------------------><
```

The selected method retrieves the text of the currently selected list entry.

Return value:

The text of the currently selected list entry, or an empty string if none is selected.

14.28. selectedIndex

```
>>--selectedIndex--------------------------------><
```

The selectedIndex method retrieves the index of the currently selected list entry. This entry is highlighted and surrounded by a dotted border. To get the selected items for a multiple selection listbox, use getListBoxData; see *getListBoxData*.

Return value:

The one-based index of the currently selected list entry, or 0 if none is selected.

14.29. selectedIndexes

```
>>--selectedIndexes------------------------------><
```

The selectedIndexes method retrieves the indexes of all items that are currently selected in the associated multiselection list box.

Return value:

A text string containing the indexes of the selected items in the list, separated by blanks.

Example:

The following example lists all items selected in the associated multiselection list box:

```
lb = self~newListBox("Offers")
if lb == .Nil then return
sit = lb~selectedItems
if sit > 0 then do
   say "You ordered:"
   sndx = lb~selectedIndexes
   do i = 1 to sit
      parse var sndx order sndx
      say "1 x" lb~getText(order)
   end
end
```

14.30. selectedItems

```
>>--selectedItems--------------------------------><
```

The selectedItems method retrieves the number of items that are currently selected in the associated multiselection list box.

Return value:

The number of selected items, or -1 if this method fails, for example if the list box is no multiselection list box.

Example:

For an example, refer to *selectedIndexes*.

14.31. selectIndex

```
>>--selectIndex(--index--)-----------------------><
```

The selectIndex method selects the list entry at the specified position. The currently selected list item is highlighted and surrounded by a dotted border. To select multiple items of a multiple selection listbox, use the *setListBoxData*() method.

Arguments:

The only argument is:

index

The index (starting with 1) of the list item to be selected. If you specify 0 for this argument, the list box must not contain any selection.

Return value:

0 if an error occurred or you specified 0 for *index* to remove the selection.

14.32. selectRange

```
>>--selectRange(--+------------+--+------------+--)-------------------------><
                  +-startIndex--+  +-,--endIndex-+
```

The selectRange method selects one or more consecutive items of the associated multiselection list box.

Arguments:

The arguments are:

startIndex
: The position of the first list item to be selected. If this argument is omitted, the selection range starts with the first item in the list.

endIndex
: The position of the last list item to be selected. If this argument is omitted, the selection range ends with the last item in the list.

If *startIndex* is equal to *endIndex*, a single item of the list is selected.

Return value:

-1 if an error occurred. This can happen, for example, if the list box is not a multiselection list box.

Example:

The following example selects the items 5 through 9:

```
lb = self~newListBox("Offers")
if lb == .Nil then return
lb~selectRange(5,9)
```

14.33. setTabulators

```
                    +-,------+
                    V        |
>>--setTabulators(----tabPos-+--)----------------><
```

The setTabulators method sets the tabulators for the associated list box control. This enables you to use items containing tab characters ("09"x), which is useful if you want to format the list in more than one column when using proportional fonts.

Arguments:

The only argument is:

tabPos
: The position or positions of the tabs relative to the left edge of the list box, in dialog units.

Return value:

1 if an error occurred.

Example:

The following example creates a list that can handle up to three tabulators in a list entry. The tabulator positions are 10, 20, and 30.

```
lb = MyDialog~newListBox(102)
if lb == .Nil then return
lb~setTabulators(10, 20, 30)
lb~add(textcol1 || "09"x || textcol2 || "09"x || textcol3 || "09"x ||,
   textcol4)
```

14.34. setWidth

```
>>--setWidth(--width--)--------------------------><
```

The setWidth method sets the internal width of the list box, in dialog units. If the internal width exceeds the width of the list box control and the list box has the HSCROLL style, the list box provides a horizontal scroll bar.

Arguments:

The only argument is:

width

The width of the list box, in dialog units.

Example:

The following example sets the internal width twice the width of the list box control:

```
lb = MyDialog~newListBox(102)
if lb == .Nil then return
lb~setWidth(lb~SizeX*2)
```

14.35. setWidthPx

```
>>--setWidthPx(--width--)------------------------><
```

The setWidthPx method sets the internal width of the list box, in pixels. If the internal width exceeds the width of the list box control and the list box has the HSCROLL style, the list box provides a horizontal scroll bar.

Arguments:

The only argument is:

width

The width of the list box, in pixels.

14.36. width

```
>>--width----------------------------------------><
```

The width method retrieves the internal width of the associated list box, in dialog units.

Return value:

The internal width of the list box, in dialog units, or 0 if an error occurred.

14.37. widthPx

```
>>--widthPx---------------------------------------><
```

The widthPx method retrieves the internal width of the associated list box, in pixels.

Return value:

The internal width of the list box, in pixels, or 0 if an error occurred.

List View Controls

A list-view control is a window that displays a collection of items, with each item consisting of an icon and a label. It provides several ways of arranging and displaying items. Refer to the **`oodlist.rex`** program in the **`samples\oodialog`** directory for an example.

The **`ListView`** class is the ooDialog interface to the Windows *List-View* control. This should not be confused with the Windows *List Box* control. The ooDialog *ListBox* class provides the interface to the Windows List Box control. In general, the List-View control is more powerful than the List Box control.

Extended Styles

The List-View control has been updated by Microsoft to include a number of *extended* styles. These styles are similar to the styles that can be used to create a ListView, (see the *createListView* method,) like the **`NOHEADER`** or **`SINGLESEL`** styles. However, the way that they are added or removed from the ListView is different. For instance, the extended styles can not be added to a ListView at the time of creation. They can only be added after the underlying Windows control has been created. For all practical purposes this means in the *initDialog* method, or at some point in the life cycle of the dialog after that.

Some of the extended styles are only available on later versions of the Windows OS. As an example, the **`LABELTIP`** and **`SIMPLESELECT`** styles are only available on Windows XP. The documentation on the extended styles will note any requirements for each style. To work with the extended List-View styles, use the *addExtendedStyle*, the *removeExtendedStyle*, or the *replaceExtendedStyle* methods.

There are a number of nuances to the behavior of the different extended styles on the different versions of Windows, far too many to try and detail in this documentation. The behavior documented here is the general behavior on Windows XP with service pack 2.

If there is a need for detailed information concerning the subtle differences in behavior on earlier versions of Windows, the programmer should consult the Windows *documentation*.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with list-view controls:

Instantiation:

Use the *newListView* method of the dialog *dialog* to retrieve a list-view object.

Dynamic Definition:

To dynamically create a list-view in the dialog template of a *UserDialog* use the *createListView* method.

Event Notification

To connect the *event* notifications sent by the underlying list-view control to a method in the Rexx dialog object use the *connectListViewEvent* method.

15.1. View Styles

List view controls can display their contents in different views. The current view is specified by the window style of the control. Additional window styles define the alignment of the items and the functionality of the list-view control. The different views are:

Icon view

Each item appears as a full-sized icon with a label below it. The user can drag the items to any location in the list view.

Small-icon view
: Each item appears as a small icon with a label to the left of it. The user can drag the items to any location.

List view
: Each item appears as a small icon with a label to the left of it. The user cannot drag the items.

Report view
: Each item appears on a separate line with information arranged in columns. The leftmost column contains the small icon and the label. All following columns contain subitems as specified by the application.

15.2. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with list-view objects, including the pertinent methods from other classes:

Table 15.1. ListView Instance Methods

Method	Description
Useful	**External Methods**
newListview	Returns a **`ListView`** object for the control with the specified ID.
createListView	Creates a list-view control in the dialog template of a *UserDialog*.
connectListViewEvent	Connects list-view event notifications to a method in the Rexx dialog object.
ListView	**Helper Classes**
LvFullRow	The **`LvFullRow`** class represents a single item and its subitems in a list-view and contains the complete set of data pertaining to that item.
LvItem	The **`LvItem`** class represents a list-view item and the data for that item.
LvSubItem	The **`LvSubItem`** class represents a single subitem of a list-view item and the data for that subitem.
Constant Methods	**Constant Methods**
Constant Methods	The **`ListView`** class provides several *constant* values through the **`::constant`**directive.
Instance Methods	**Instance Methods**
add	Adds a new item to this list-view or defines the text for a subitem.
addExtendedStyle	Adds one or more of the extended list-view styles to this list-view control.
addFullRow	Adds a new item to this list-view at the end of the list using a *LvFullRow* object.
addRow	Adds a new item to this list-view.
addRowFromArray	Inserts an item into this list-view from an array of values. The string value of each item in the array is used as the text for the item and subitem(s).
addStyle	Adds new window styles to this list-view control.
alignLeft	Aligns items along the left window border.
alignTop	Aligns items along the upper window border.
arrange	Aligns items according to the current alignment style of the list-view control.

Method	Description
bkColor	Retrieves the background color for this list-view control.
bkColor=	Sets the background color of this list-view control.
check	Sets the check mark for a single list-view item.
checkAll	Sets the check mark for every list-view item.
columnInfo	Retrieves information for a column of this list-view control in a **`Stem`** object. **(Inaccurate.)**
columnWidth	Retrieves the width, in dialog units, of a column in this list-view when it is has the list or report view style. **(Inaccurate.)**
columnWidthPX	Retrieves the width, in pixels, of a column in this list-view when it is has the list or report view style.
delete	Removes an item from this list-view control.
deleteAll	Removes all times from this list-view control.
deleteColumn	Removes a column from this list-view control.
deselect	Deselects an item in this list-view.
deselectAll	Deselects all items in this list-view.
dropHighlighted	Retrieves the item that is highlighted as a drag-and-drop target.
edit	Begins the editing of the text of the specified list-view item.
endEdit	Cancels the editing of the list-view item being edited.
ensureVisible	Ensures that a list-view item is entirely or partially visible by scrolling this list-view control, if necessary.
find	Searches for a list-view item containing the specified text.
findNearestXY	Searches, in the specified direction, for the item nearest to the specified position.
findPartial	Searches for a list-view item beginning with the specified text.
firstVisible	Retrieves the index of the topmost visible item when in this list-view is in list or report view.
fixFullRowColumns	A convenience method used to adjust the subitem columns in the *LvFullRow* objects used for *internal* sorting when a column has been deleted or inserted in this list-view.
focus	Assigns the focus to the specified item,
focused	Retrieves the index of the item that currently has the focus.
getCheck	Determines if the item at the specified index has a checked, check box.
getColumnInfo	Retrieves the attributes of this list-view column in a **`Directory`** object.
getColumnCount	Returns the number of columns in the list-view control when it has the report style.
getColumnOrder	Retrieves the current column order of the list-view when it has the report style.
getColumnText	Retrieves the column header text for this list-view when it is in report view.
getExtendedStyle	Returns the extended styles of this list-view control.
getExtendedStyleRaw	Returns the extended styles of this list-view control as the numeric value of the extended style flags.

Method	Description
getFullRow	Returns a *LvFullRow* object for the specified item in this list view.
getHoverTime	Retrieves the current hover time in milliseconds.
getImageList	Retrieves the current image list for the image list type specified.
getItem	Returns some, or all, of the information this list-view maintains on the specified item. The information is returned as a *LvItem* object. The item is specified by either a **`LvItem`** object or by the item's index.
getItemData	Retrieves the item data associated with the specified list-view item.
getItemInfo	Retrieves the attributes of a list-view item in a **`Directory`** object.
getItemPos	Retrieves the client area coordinates of an item in this list-view control as a **`Point`** object.
getItemRect	Gets the bounding *rectangle* for all or part of an item in the current view.
getSubitem	Receives some, or all, of the information of a subitem of a list-view item. The information is returned as a *LvSubItem* object.
getSubitemRect	Gets the bounding *rectangle* for all or part of a subitem in the current view of this list-view control.
getToolTips	Retrieves the child *ToolTip* control used by this list-view.
getView	Gets the current view of this list-view.
hasCheckBoxes	Determines if this list-view control has the CHECKBOXES extended list-view style.
hitTestInfo	Determines which list-view item, if any, is at the specified position.
insert	Inserts a new item into this list-view, or inserts a new subitem into in an existing list-view item.
insertColumn	Inserts a new column into this list-view control using dialog units for the width. **(Inaccurate.)**
insertColumnPX	Inserts a new column into this list-view specifying the column width in pixels.
insertFullRow	Adds a new item to this list-view at the position specified using a *LvFullRow* object.
isChecked	Determines if a check box for this list-view item is checked or not.
itemInfo	Retrieves the attributes of this list-view item in a **`Stem`** object.
itemPos	Retrieves the client area coordinates of an item in this list-view control as a **`String`** object.
items	Retrieves the number of items in this list-view control.
itemState	Retrieves the state of this list-view item, focused, selected, etc.
itemText	Retrieves the text of this list-view item or subitem.
itemsPerPage	Calculates the number of items that vertically fit the visible area of this list-view control that has the report or list style.
last	Retrieves the index of the last item in this list-view control.
lastSelected	Returns the index of the selected item with the highest index in this list-view control.
modify	Modifies the text or icon for an item, or the text for a subitem, for this list-view item

Method	Description
modifyColumn	Sets new attributes for a column of this list-view control when it has the report or list style. **(Inaccurate.)**
modifyColumnPX	Sets new attributes for a column of this list-view control when it has the report or list style.
modifyFullRow	Modifies the information for an item, and all its subitems, in a list-view using a *LvFullRow* object.
modifyItem	Modifies some or all of the data this list-view maintains for an item using a *LvItem* object.
modifySubitem	Modifies some or all of the data this list-view maintains for a subitem of an item using a *LvSubItem* object.
next	Retrieves the item that follows, or is to the right of, the specified item.
nextLeft	Retrieves the item to the left of the specified item.
nextRight	Retrieves the item to the right of the specified item.
nextSelected	Retrieves the next selected item that follows, or is to the right of, the specified item.
prepare4nItems	Informs the list-view control to prepare for the addition of a large number of items.
prependFullRow	Adds an item to this list-view at the first position in the list using a *LvFullRow* object.
previous	Retrieves the index of the item that precedes, or is to the left of, the specified item.
previousSelected	Retrieves the first selected item that precedes, or is to the left of, the specified item.
redrawItems	Forces this list-view control to redraw a range of items.
removeItemData	Removes and returns the item data associated with the specified list-view item.
removeExtendedStyle	Removes one or more extended styles of this list-view control.
removeStyle	Removes one or more window styles of this list-view control.
replaceExtendedStyle	Removes some or all of the list-view control's extended styles and adds new extended styles
replaceStyle	Removes some of the window styles of this list-view control and sets new styles.
scroll	Scrolls the content of a list view control.
select	Selects an item.
selected	Returns the index of the first selected item.
selectedItems	Determines the count of selected items in this list-view control.
setColumnOrder	Changes the column order of this list-view control to some other order.
setColumnWidth	Sets the width, in dialog units, of a column in this list-view control that has the report or list style. **(Inaccurate.)**
setColumnWidthPX	Sets the width, in pixels, of a column in this list-view control that has the report or list style.
setFullRowText	Modifies all, or some, of the text for a list-view item and its subitems using a *LvFullRow* object.

Method	Description
setHoverTime	Used to change the hover time.
setImageList	Assigns, or removes, an image list for the list-view control.
setItemData	Sets the item data associated for the specified list-view item.
setItemPos	Moves an item to a specified position in this list-view control that has the icon or small-icon style.
setItemState	Sets the state of this list-view item.
setItemText	Changes the text of an item, or the text of a subitem, of this list-view item.
setView	Sets the view of this list-view.
setToolTips	Sets the child *ToolTip* control used by this tree-view.
smallSpacing	Determines the spacing between items in this list-view control when it has the small-icon style.
snapToGrid	Snaps all icons to the nearest grid position.
sortItems	The *sortItems* method causes the list-view control to sort its items using the specified comparison method in the Rexx dialog.
spacing	Determines the spacing between items in an icon list-view control.
stringWidth	Determines the width, in dialog units, of a specified string using the current font of the list-view control. **(Inaccurate.)**
stringWidthPX	Determines the width, in pixels, of a specified string using the current font of the list-view control.
textBkColor	Retrieves the background color of the text in this list-view control.
textBkColor=	Sets the background color for the text in this list-view control.
textColor	Retrieves the text color of this list-view control.
textColor	Sets the text color of this list-view control.
uncheck	Removes the check mark for the specified list-view item.
uncheckAll	Clears the check mark for every list-view item.
updateItem	Updates this list-view item.

15.3. Constant Methods

The **`ListView`** class provides the following *constant* values through the use of the **`::constant`** directive.

Table 15.2. ListView Class Constant Reference

Constant Symbol	Description
GROUPIDCALLBACK	A group ID index value that indicates the list-view control should send the GETDISPINFO notification message to retrieve the item's group index.
GROUPIDNONE	A group ID value that indicates the list-view item does not belong to a group.
IMAGECALLBACK	An index value for the item's icon in the control's image list that indicates the list-view control should send the GETDISPINFO notification message to retrieve the item's icon index when it needs to display the image.

Constant Symbol	Description
IMAGENONE	An index value for the item's icon in the control's image list that indicates the list-view item does not have an icon.
LVCFMT_FILL	A column format flag, fill the remainder of the tile area. Might have a title. Only valid in extended tile view. The LVCFMT_FILL flag forces the column to fill the remainder of the tile area. **Requires Windows Vista or later**.
LVCFMT_LINE_BREAK	A column format flag, move to the top of the next list of columns. Only valid in extend tile view, The LVCFMT_LINE_BREAK flag forces the column to wrap to the top of the next list of columns. **Requires Windows Vista or later**.
LVCFMT_NO_TITLE	A column format flag, this subitem does not have a title. Only valid in extended tile view. The LVCFMT_NO_TITLE flag indicates that the column should not display a title. **Requires Windows Vista or later**.
LVCFMT_TILE_PLACEMENTMASK	A column format flag, this is a combination of the LVCFMT_LINE_BREAK and LVCFMT_FILL constants. Only valid in extended tile view. **Requires Windows Vista or later**.
LVCFMT_WRAP	A column format flag, this subitem can be wrapped. Only valid in extended tile view. The LVCFMT_WRAP flag allows the column to wrap within the remaining space in its list of columns. **Requires Windows Vista or later**.

15.4. newListView (dialog object method)

List view objects can not be instantiated by the programmer from Rexx code. Rather a list-view object is obtained by using the *newListView* method of the *dialog* object. The syntax is:

```
>>--newListView(--id--)--------------------------><
```

15.5. createListView (UserDialog method)

A list-view object can be created in the dialog template for a *UserDialog* dialog through the *createListView* method. The basic syntax is:

```
>>--createListView(-id-,--x-,--y-,--cx-,--cy-+---------+--+------------+--)----><
                                             +-,-style-+  +-,-attrName-+
```

15.6. connectListViewEvent (dialog object method)

To connect event notifications from a list-view object use the *connectListViewEvent* method of the *dialog* object. The basic syntax is:

```
>>-connectListViewEvent(--id--,--event--+---------------+--)----><
                                        +-,--methodName-+
```

15.7. add

```
          +-------+
          V       |
>>--add(----+-----+--text--+--------+--)---------><
            +--,--+        +-,-icon-+
```

The *add* method adds a new item to a list-view, or defines the text for a subitem. It can be used to fill a list-view sequentially.

Remember that in a list-view, items and columns use a 0-based index. Subitems use a 1-based index.

The number of commas preceding the *text* argument determines whether a new item is added to the list-view or the text for a subitem is being defined. When there is no comma preceding *text* then a new item is added to the list-view.

When 1 or more commas precede *text*, then no item is added to the list-view. Rather, the text for a subitem is defined. When 1 comma precedes *text*, then the text for subitem 1 is being defined. When 2 commas precedes the *text* argument, then the text for subitem 2 is being defined, and so on.

When an item is added, it is always added directly *after* the last item to be placed in the list-view. Or as item 0 if the list-view currently has no items. When the text of a subitem is being defined, then the text is defined for the subitem of last item placed in the list-view.

Arguments:

The arguments are:

, [optional]
: As explained above, this argument determines if an item is added to the list-view, or if the text of a subitem is defined.

text [required]
: The text for the item label, or the text for the subitem of a item, depending on whether an item is being added or the text for a subitem is being defined.

icon [optional]
: Specifies the index within the image list for the icon for this item. Image lists are assigned to a list-view by using the *setImageList*() method. With the **`setImageList`**() method, use the LVSIL_NORMAL flag for the icon view icons and the LVSIL_SMALL flag for the list, report, and small-icon view icons.

: Use -1 or simply omit this argument to indicate there is no icon for the item being added. This argument is ignored completely when the text for a subitem is being defined.

Example:

The following example adds three columns and two items to a report list control. To get the following result:

First Name	Last Name	Age
Mike	Miller	30
Sue	Thaxtor	29

using the *add* method you must specify something similar to the following:

```
::method initDialog
```

```
list = self~newListView(IDC_LIST_REP)
if list \= .nil then do
  imageList = .ImageList~create(.Size~new(16), COLOR24, 9, 0)
  imageList~add(.Image~getImage("E:\oodlist\oodlist.BMP"))
  list~setImageList(imageList, SMALL)

  list~insertColumn(0,"First Name",50)
  list~insertColumn(1,"Last Name",50)
  list~insertColumn(2,"Age",50)

  -- 3 is the index in the image list for the icon for item 0
  list~add("Mike", 3)
  list~add(,"Miller")
  list~add(, ,"30")

  -- 0 is the index in the image list for the icon for item 1
  list~~add("Sue", 0)~~add(,"Thaxtor")~~add(, ,"29")
end
```

15.8. addExtendedStyle

```
>>-addExtendedStyle(--style--)--------------------><
```

The *addExtendedStyle* method adds one or more of the *extended* list-view styles to a list-view. These styles can not be added until the underlying Windows list-view control has been created. They can be added in the *initDialog* method, or any time thereafter.

Arguments:

The only argument is:

style [required]

The extended style to be added. A list of 1 or more of the following keywords separated by spaces, case is not significant. Invalid words in the list are ignored. If there are no valid keywords at all in the list, then an error is returned:

BORDERSELECT	HEADERDRAGDROP	SIMPLESELECT
CHECKBOXES	INFOTIP	SUBITEMIMAGES
DOUBLEBUFFER	LABELTIP	TRACKSELECT
FLATSB	MULTIWORKAREAS	TWOCLICKACTIVATE
FULLROWSELECT	ONECLICKACTIVATE	UNDERLINECOLD
GRIDLINES	REGIONAL	UNDERLINEHOT

BORDERSELECT

When an item is selected the border color of the item changes rather than the item being highlighted.

CHECKBOXES

Check boxes are visible and functional in all list-view modes except tile mode.

DOUBLEBUFFER

Windows XP or later only. The list-view control is painted using double buffering. This reduces flicker.

FLATSB

Enables flat scroll bars.

FULLROWSELECT
: Report view only. When a row is selected, the whole row is highlighted rather than just the first column.

GRIDLINES
: Report view only. Gridlines are drawn around all items and sub-items. This gives a spreadsheet-like appearance to the list-view control.

HEADERDRAGDROP
: Report view only. The user can drag and drop the headers to rearrange the columns.

INFOTIP
: With this style the list-view control sends the *GETINFOTIP* notification to the dialog before displaying an item's tooltip. This allows the program to add, enhance, or change the text displayed in the tip.

LABELTIP
: Windows XP or later only. In any list-view mode, if an item's label is only partially visible, the complete label will be displayed in a fashion similar to a tool tip.

MULTIWORKAREAS
: The current version of ooDialog can not take advantage of this style. It is planned for a future enhancement.

ONECLICKACTIVATE
: Enables the list-view control to send an item activate notification when the user clicks one time on an item. It also enables hot tracking in the list-view control. Both the underline cold and the underline hot styles need either one click activate or two click activate styles to be enabled.

REGIONAL
: Icon view only. Sets the window region to only include an item's icon and text.

SIMPLESELECT
: Windows XP and later only. Icon view only. Moves the state image to the top right of the icon image. If the user changes the state using the space bar, all the selected items cycle, not just the item with the focus.

SUBITEMIMAGES
: Report view only. Allows images to be displayed for subitems.

TRACKSELECT
: Enables hot track selection. The user can select an item by leaving the mouse over an item for a certain period of time, the "hover" time. This period of time can be changed from the default by using the *setHoverTime* method.

TWOCLICKACTIVATE
: Enables the list-view control to send an item activate notification when the user clicks on an item after it has been selected. When the user clicks on an item it will be selected. At that point, if the user clicks on the item a second time, the item will be activated. Note that this is different from double-clicking to activate. Any amount of time can pass between the click that selects the item and the click that activates the item. (Provided the item remains selected of course.)

It also works with track select. If track select is enabled, the user can select an item by pausing the mouse over the item. At that point, if the user then clicks on the item it will be activated.

UNDERLINECOLD

Underlining gives the user a visual clue that a single click on an item will activate it. Underlining requires either ONECLICKACTIVATE or TWOCLICKACTIVE to also be enabled. An item becomes hot when the mouse is over it. Underline cold underlines all the cold items when a single click will activate them. When one click activate is enabled, this would then be all items. When two click activate is enabled this would then be only those items that were selected.

UNDERLINEHOT

The underline hot style is similar to the underline cold style. See that style above for the explanation of some of the terms. With underline hot, the item is underlined when the mouse is over it, if one click will activate the item.

Return value:

The return values are:

0

Success.

-1

An (internal) problem with the control's window handle.

-3

The style argument did not contain any of the extended style keywords.

Example:

The following example shows how to initially add the extended styles to a list-view control. This is done in the *initDialog* method, which executes after the underlying Windows dialog has been created, but before it is displayed on the screen. This is the earliest point in time that the extended style methods can be used. The list-view is in the report view and will have check boxes and grid lines. The user will be able to use drag and drop to rearrange the columns. When a row is selected, the full row will be highlighted rather than just the first column.

```
::method initDialog
  expose listView

  ...
  listView = self~newListView(IDC_LIST)
  if listView == .nil then return

  listView~addExtendedStyle("CHECKBOXES GRIDLINES FULLROWSELECT HEADERDRAGDROP")
  ...
```

15.9. addFullRow

```
>>--addFullRow(--row--)--------------------------><
```

Add an item to the list-view at the end of the list using a *LvFullRow* object.

Arguments:

The single argument is:

row [required]

A *LvFullRow* object that specifies the item to be added to the list-view.

Return value:

On success, returns the zero-based index of the added row. On error, -1 is returned.

Remarks:

When a full row object is used to add an item to the list view, all the subitems are also added. The item is always added as the last item in the list. Because of this, the item index and subitem index values in the full row object are ignored. After the item is added, the item and subitem index are updated to the correct value of the newly added item.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows part of an initDialog() method. The private method createRows() returns an array of **`LvFullRow`** objects, which are then used to populate the list-view:

```
::method initDialog
  expose list

  list = self~newListView(IDC_LV_EMPLOYEES)

  ...

  rows = self~createRows
  do r over rows
    list~addFullRow(r)
  end
```

15.10. addRow

```
                                     +-,----+
                                     V      |
>>--addRow(--+------+--,--------+--+--------+--)-><
             +-item-+  +-,-icon-+  +-,-text-+
```

The *addRow* method adds a new item to a list-view. This method is most useful when adding items that have subitems to the list-view. But, despite its name, it can be used to add any item, with or without subitems, to a list-view.

Remember that in a list-view, items and columns use a 0-based index. Subitems use a 1-based index.

Arguments:

The arguments are:

item [optional]

The index of the item. If you omit this argument, the item is added immediately after the last item added or inserted in the list-view. That is, the index defaults to the index of the last item added plus 1. If there are no items in the list-view then the item is added at index 0.

icon [optional]

The index within the image list for the icon for this item. Image lists are assigned to a list-view by using the *setImageList*() method. With the *setImageList* method, use the LVSIL_NORMAL flag for the icon view icons and the LVSIL_SMALL flag for the list, report, and small-icon views icons. If this argument is omitted, no icon index is assigned to the itme

text [optional]

Any number of text strings. The first text string is used for the label of the item. Successive strings are used for the text of subitems of the item. For example, the text of the second string is used for the text of subitem 1. The text of the third string is used for the text of subitem 2, and so on. If the first string is omitted, then the empty string is used for the item label. If other strings are omitted, than no text is set for the corresponding subitem. If you specify more text entries than there are columns, the extra entries are ignored.

Return value:

The index of the new item, or -1 on error.

Example:

The following example adds three items to a report list with two columns:

```
::method InitList
  lv = self~newListView(IDC_LV_CONTACTS)
  lv~addRow( , ,"Mike","Miller")
  lv~addRow( , ,"Sue","Muller")
  lv~addRow( , ,"Chris","Watson")
```

15.11. addRowFromArray

```
>>--addRowFromArray(--data--+--------------+--+-------------+--)--------------><
                            +-,-itemIndex--+  +-,-strIfNil--+
```

Inserts an item into this list-view from an array of values. The string value of each item in the array is used as the text for the item and subitem(s).

Arguments:

The arguments are:

data [required]

An array object. The string value of each item in the array is used as the text for the item and subitems of the list-view item. The item at index 1 of the array is used as the text for the list-view item. The item at index 2 is used as the text for the first subitem, the item at index 3 is used as the text for subitem 2, etc.. The array can not be sparse

itemIndex [optional]

The item index to use for the for the list-view item being added. If omitted, the new item is added after the last inserted item in the list-view.

strIfNil [optional]

If the item at any index in *data* array is the **`.nil`** object, then the text for the item or subitem corresponding to that index will be the string value of the **`.nil`** object. Which is *the NIL object*. The *strIfNil* argument can be used to specify text to use instead of *the NIL object*.

Return value:

Returns the index of the newly added list-view item on success. On error, -1 is returned.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example queries an ooSQLite database and then populates the list-view items with the returned result set:

```
dbConn = .ooSQLiteConnection~new(dbName, .ooSQLite~OPEN_READWRITE)

sql  = 'SELECT * FROM foods ORDER BY name;'
data = dbConn~exec(sql, .true)
dbConn~close

self~insertListViewColumns(list, data[1])
do i = 2 to data~items
    list~addRowFromArray(data[i], , 'null')
end
```

15.12. addStyle

```
                              +-------------------+
                              V                   |
>>-aListControl~addStyle(--"----+-ICON----------+-+--"--)-------><
                                +-SMALLICON-----+
                                +-LIST----------+
                                +-REPORT--------+
                                +-ALIGNLEFT-----+
                                +-ALIGNTOP------+
                                +-AUTOARRANGE---+
                                +-ASCENDING-----+
                                +-DESCENDING----+
                                +-EDIT----------+
                                +-HSCROLL-------+
                                +-VSCROLL-------+
                                +-NOSCROLL------+
                                +-NOHEADER------+
                                +-NOSORTHEADER--+
                                +-NOWRAP--------+
                                +-SINGLESEL-----+
                                +-SHOWSELALWAYS-+
                                +-SHAREIMAGES---+
                                +-GROUP---------+
                                +-HIDDEN--------+
                                +-NOTAB---------+
                                +-NOBORDER------+
```

The addStyle method adds new window styles to a list-view control.

Arguments:

The only argument is:

style

The window styles to be added, which is one or more of the styles listed in the syntax diagram, separated by blanks. For an explanation of the different styles, refer to *createListView*.

Return value:

0 if this method fails.

15.13. alignLeft

```
>>--alignLeft------------------------------------><
```

The alignLeft method aligns items along the left window border.

Return value:

0
: The items were aligned.

1
: For all other cases.

15.14. alignTop

```
>>--alignTop-------------------------------------><
```

The alignTop method aligns items along the upper window border.

Return value:

0
: The items were aligned.

1
: For all other cases.

15.15. arrange

```
>>--arrange--------------------------------------><
```

The arrange method aligns items according to the current alignment style of the list-view control.

Return value:

0
: The items were aligned.

1
: For all other cases.

15.16. bkColor

```
>>--bkColor--------------------------------------><
```

The bkColor method retrieves the background color for a list-view control.

Return value:

The *color palette* index specifier (0 to 18).

15.17. bkColor=

```
>>--bkColor = color------------------------------><
```

The bkColor= method sets the background color of a list-view control.

Arguments:

The only argument is:

color

The new background color. Specify the *color palette* index specifier (0 to 18).

Example:

The following example sets the background color of a list control to yellow:

```
::method Yellow
  curList = self~newListView(104)
  curList~bkColor = 15
  curList~update
```

15.18. check

```
>>-aListControl~check(--index--)-----------------><
```

The **check** method sets the check mark for a list-view item. This method is only valid for list-view controls that have the check boxes extended style. (See the *addExtendedStyle* method for a description of the check boxes extended style.)

Arguments:

The only argument is:

index

The index of the item for which to set the check mark.

Return value:

The return values are:

0

Success.

-1

An (internal) problem with the control's window handle.

-2

The list-view control does not have the check boxes extended style.

-3

The index is invalid.

Example:

The following example is from a mailing list application. The user selects which entries to include in any mailing. All items in a list-view that have their check boxes checked are included in the mailing. The application has a feature that allows the user to automatically select all items with a specified zip code.

```
::method onUseZip

  text = self~newEdit(IDC_EDIT_ZIPCODE)~getText
  if self~validateZip(text) then do
     list = self~newListView(IDC_LIST)
     do i = 0 to addresses~items - 1
       infoTable = address[i + 1]
       if infoTable['zip'] == text then list~check(i)
     end
  end
```

15.19. checkAll

```
>>-aListControl~CheckAll-------------------------><
```

The **`checkAll`** method sets the check mark for every list-view item. This method is only valid for list-view controls that have the check boxes extended style. (See the *addExtendedStyle* method for a description of the check boxes extended style.)

Arguments:

This method takes no arguments.

Return value:

The return values are:

0

Success.

-1

An (internal) problem with the control's window handle.

-2

The list-view control does not have the check boxes extended style.

Example:

The following example is from a program that has a push button that allows the user to check all the check boxes:

```
::method onCheckAll

  list = self~newListView(IDC_LIST_REPUBLICANS)
  list~checkAll
```

15.20. columnInfo

```
>>-aListControl~columnInfo(--column--)-----------><
```

The columnInfo method retrieves the attributes of a column of a list-view control.

Arguments:

The only argument is:

column

The number of the column of which the attributes are to be retrieved. 0 is the first column.

Return value:

A compound variable that stores the attributes of the item. If attributes for a column are not available or an error occurs, then the RetStem.!WIDTH variable will be set to 0.

RetStem.!TEXT
: The heading of the column.

RetStem.!COLUMN
: The column number.

RetStem.!WIDTH
: The width of the column.

RetStem.!ALIGN
: The alignment of the column: "LEFT", "RIGHT", or "CENTER".

15.21. columnWidth

```
>>-aListControl~columnWidth(--column--)----------><
```

The columnWidth method retrieves the width of a column in a report or list view.

Arguments:

The only argument is:

column
: The number of the column of which the width is to be retrieved. 0 is the first column.

Return value:

The column width, or -1 if you did not specify *column*, or 0 in all other cases.

Example:

The following example displays the column width in an information box when the column is clicked on:

```
::method onColumnClick
  use arg id, column
  listView = self~newListView(102)
  call infoDialog listView~columnWidth(column)
```

15.22. columnWidthPX

```
>>--columnWidthPX(--colIndex--)------------------><
```

Gets the width, in pixels, of the specified column in a list-view control when it has the list or report view styles.

Arguments:

The single argument is:

colIndex [required]
: The zero-based index of the column to query.

Return value:

The width of the column, in pixels, on success or zero on error.

Remarks:

The *colIndex* argument is ignored when the list-view has the list style.

15.23. delete

```
>>--delete(--index--)----------------------------><
```

Deletes the specified item from the list-view control.

Arguments:

The single argument is:

index [required]

The zero-based index of the item to delete.

Return value:

Returns 0 on success, 1 on error.

Example:

This example deletes the selected item.

```
::method deleteSelectedItem
    list = self~newListView(IDC_LV_RACECARS)

    index = list~selected
    if index <> -1 then
        list~delete(index)
```

15.24. deleteAll

```
>>--deleteAll------------------------------------><
```

Deletes all items from the list-view control.

Arguments:

This method has no arguments.

Return value:

Returns 0 on success, 1 on error.

15.25. deleteColumn

```
>>--deleteColumn(--colIndex--+------------------+--)------------><
                             +-,-adjustFullRows-+
```

Deletes the specified column in this list-view.

Arguments:

The arguments are:

colIndex [required]

The zero-based index of the column to delete.

adjustFullRows [optional]

If true attempts to fix up the *LvFullRow* objects assigned to the item *data* of each list-view item. Does nothing if false. The default is false. This argument is only used to support *internal* sorting. If the application is not using internal sorting then this argument should just be ignored.

Return value:

Returns 0 on success, 1 on error.

Remarks:

MSDN says column 0 can not be deleted. If column 0 must be deleted, MSDN suggests inserting a dummy column at index 0 and then deleting column 1.

However, this is not true, deleting column 0 will succeed if all the other columns are deleted first. What can not be deleted is the list-view item information for column 0. When a column for a subitem is deleted, the subitem information is deleted. But, when column 0 is deleted the item information is not deleted. If, after deleting column 0, the application switches the view from report to one of the other views, like icon, the information for each item will display. In report view, nothing will show if all columns have been deleted.

Only use the *adjustFullRows* argument to *adjust* the **`LvFullRow`** objects if every list-view item has a **`LvFullRow`** item assigned as the item data, and the programmer has kept the full rows consistent with changes to the column count of this list-view. If these conditions are not met, this method will return an error code, even if the column was deleted successfully. However, no harm is done, it will just make it hard to tell between a successful execution of the method and an error.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example deletes the column for the first subitem of the list-view items:

```
list = self~newListView(IDC_LV_CONTACTS)
list~deleteColumn(1)
```

15.26. deselect

```
>>-aListControl~deselect(--item--)---------------><
```

The deselect method deselects an item.

Arguments:

The only argument is:

item

The number of the item.

Return value:

0
: The item was selected.

-1
: You did not specify *item*.

1
: For all other cases.

15.27. deselectAll

```
>>--deselectAll----------------------------------><
```

Deselects all items in the list-view.

Arguments:

This method has not arguments.

Return value:

Returns the number of items deselected.

Example:

This example deselects any selected items in the list-view.

```
  self~newListView(IDC_LV_ADDRESSES)~deselectAll
```

15.28. dropHighlighted

```
>>-aListControl~DropHighlighted------------------><
```

The dropHighlighted method retrieves the item that is highlighted as a drag-and-drop target.

Return value:

The number of the selected item, or -1 in all other cases.

15.29. edit

```
>>-aListControl~edit(--item--)-------------------><
```

The edit method begins editing of the text of the specified list-view item.

Arguments:

The only argument is:

item
: The number of the item.

Return value:

The handle of the edit control used to edit the item text, or 0 in all other cases.

15.30. endEdit

```
>>-aListControl~EndEdit--------------------------><
```

The endEdit method cancels editing of the list view item that is being edited.

15.31. ensureVisible

```
                                         +-0-+
>>-aListControl~ensureVisible(--item--,--+---+--)---------------><
                                         +-1-+
```

The ensureVisible method ensures that a list-view item is entirely or partially visible by scrolling the list-view control, if necessary.

Arguments:

The arguments are:

item

The number of the item visible.

partial

Specifies whether the item must be entirely visible:

1

The list-view control is not scrolled if the item is at least partially visible.

0

The list-view control is scrolled if the item is only partially visible. This is the default.

Return value:

0

The item is visible.

-1

You did not specify *item*.

1

For all other cases.

15.32. find

```
                                +- -1--+     +-0-+
>>-aListControl~find(--text--,--+------+--,--+---+--)-----------><
                                +-item-+     +-1-+
```

The find method searches for a list-view item containing *text*. The text of this item must exactly match *text*.

Arguments:

The arguments are:

text

The text of the item to be searched for.

item

Specify the number of the item at which the search is to be started. Specify -1 or omit this argument to start the search at the beginning.

wrap

Specify 1 if the search is to be continued at the beginning if no match is found. Specify 0 or omit this argument if the search is to stop at the end of the list.

Return value:

The number of the item, or -1 in all other cases.

15.33. findNearestXY

```
>>-aListControl~findNearestXY(--x--,--y--+--------------------+--)-------------><
                                         |       +-DOWN--+    |
                                         +-,--"--+-UP----+--"-+
                                                 +-LEFT--+
                                                 +-RIGHT-+
```

The findNearestXY method searches, in the specified direction, for the item nearest to the specified position.

Arguments:

The arguments are:

x

The x-coordinate of the position at which the search is to be started.

y

The y-coordinate of the position at which the search is to be started.

direction

The direction in which the search should proceed.

Return value:

The index of the item, or -1 in all other cases.

15.34. findPartial

```
                                       +- -1--+     +-0-+
>>-aListControl~findPartial(--text--,--+------+--,--+---+--)----><
                                       +-item-+     +-1-+
```

The findPartial method searches for a list-view item containing *text*. An item matches if its text begins with *text*.

Arguments:

The arguments are:

text

The text of the item to be searched for.

item

Specify the number of the item at which the search is to be started. Specify -1 or omit this argument to start the search at the beginning.

wrap

Specify 1 if the search is to be continued at the beginning if no match is found. Specify 0 or omit this argument if the search is to stop at the end of the list.

Return value:

The number of the item, or -1 in all other cases.

15.35. firstVisible

```
>>--firstVisible---------------------------------><
```

Retrieves the index of the topmost visible item when in this list-view is in list or report view.

Arguments:

There are no arguments for this method.

Return value:

Returns the zero-based index of the topmost visible item when the list-view is in list or report view. Returns 0 if the list-view is in icon or small icon view.

Remarks:

There is no way to distinguish between a return of 0 because the list-view is not in list or report view and a 0 because the topmost visible item is index 0 when the list-view is in list or report view.

15.36. fixFullRowColumns

```
>>--fixFullRowColumns(--colIndex--+-------------+--)------------><
                                  +-,-inserted--+
```

A convenience method used to adjust the subitem columns in the *LvFullRow* objects used for *internal* sorting when a column has been deleted or inserted in this list-view.

Arguments:

The arguments are:

colIndex [required]

The one-based index of the *subitem* column that was inserted or deleted.

inserted [optional]

True or false to specify if the column was inserted or removed. The default is false, the column was deleted. Set this argument to true to specify the column was inserted.

Return value:

Returns true on success, false on error.

Remarks:

As explained in the *section* on internal sorting considerations, the ooDialog framework does its best to keep the full row objects assigned as the *item* data of each list-view item in sync with the actual item. However, when a column is deleted or inserted, the programmer needs to take some action to alert the framework that the full row objects need to be updated. The *fixFullRowColumns* is provided as one option the programmer can use to take action. The other options are to manually update the item data of each list-view item, or to use the *adjustFullRows* argument of the *deleteColumn* or *insertColumn* methods.

The *fixFullRowColumns* method has to be used after each individual delete or insert of a column. I.e., the programmer can not delete 2 rows and then call *fixFullRowColumns* twice in a row. The *colIndex* argument value must be within a valid range. It can not be 0, or greater than the count of current columns plus 1.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example uses the *fixFullRowColumns* method to notify the ooDialog framework that a column has been deleted in the list-view. The framework then updates all the **`LvFullRow`** objects assigned as the item data of each list-view item:

```
list = self~newListView(IDC_LV_CONTACTS)
list~deleteColumn(1)
list~fixFullRowColumns(1)
```

15.37. focus

```
>>-aListControl~focus(--item--)------------------><
```

The focus method assigns the focus to the specified item, which is then surrounded by the standard focus rectangle. Although more than one item can be selected, only one item can have the focus.

Arguments:

The only argument is:

item

The number of the item to receive the focus.

Return value:

0

The specified item received the focus.

-1

You did not specify *item*.

1

For all other cases.

15.38. focused

```
>>-aListControl~Focused--------------------------><
```

The focused method retrieves the number of the item that has currently the focus.

Return value:

The number of the item with the focus, or -1 in all other cases.

15.39. getCheck

```
>>-aListControl~getCheck(--index--)--------------><
```

The **`getCheck`** method determines if the item at the specified list index has a checked check box. This method will return a code indicating that the list-view control does not have the check boxes extended style, if appropriate. (See the *addExtendedStyle* method for a description of the check boxes extended style.)

Arguments:

The only argument is:

index

The index of the item to query.

Return value:

The return values are:

1

The item has a check box that is checked.

0

The item has a check box, and it is not checked.

-1

An (internal) problem with the control's window handle.

-2

The list-view control does not have the check boxes extended style.

-3

The index is invalid.

Example:

The following example could be used as a shortcut method to determine if a list-view control currently has the check box extended style.

```
...
list = self~newListView(IDC_LV_DOCTORS)
if list~getCheck(0) == -2 then
  return   -- The list-view does not currently have the check box style in effect

do i = 0 to list~items - 1
  ...
end
```

15.40. getColumnCount

```
>>--getColumnCount-------------------------------><
```

This method returns the number of columns in the list-view control when it is in report view.

Arguments:

There are no arguments.

Return value:

This method returns the number of columns, or -1 on some error. An error is unlikely.

Example:

This method is straight forward to use:

```
colCount = list~getColumnCount
```

15.41. getColumnInfo

```
>>--getColumnInfo(--index--,--directory--)-------><
```

Retrieves information for the column specified in a **`Directory`** object.

Arguments:

The arguments are:

index [required]

The zero-based index of the column to retrieve information for. Columns are numbered from the left to the right.

directory [required]

A **`Directory`** object whose indexes will be filled with the returned column information. On success, the **`Directory`** object will have the following indexes with the corresponding information:

text

The text (label) for the column.

subitem

The index of the subitem associated with the column.

width

The width, in pixels, of the column.

fmt

A series of blank separated key words specifying the format for the column. See the remarks.

Return value:

Returns **`.true`** on success and **`.false`** on error.

Remarks:

No other indexes in the *directory* object will be touched other than those noted. On error, the value of the noted indexes is undefined.

The **`fmt`** index, at this time, will contain exactly one keyword, either LEFT, RIGHT, or CENTER denoting the alignment of the text for the column. However, the underlying list-view control has a number of other format attributes. These other attributes are currently ignored by the ooDialog framework. Future versions of ooDialog may be enhanced to include more keywords.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

This example displays information about the column when the user clicks on it:

```
::method onColumnClick
  use arg id, column
  listView = self~newListView(IDC_LV_PROPERTIES)

  d = .Directory~new
  if listView~getColumnInfo(column, d) then do

    msg = "Column Title : " d~text    || .endOfLine || -
          "Column Number: " d~subitem || .endOfLine || -
          "Column Width : " d~width   || .endOfLine || -
          "Alignment    : " d~fmt

    j = MessageDialog(msg, self~hwnd, "List View Column Information")
  end
  else do
    j = MessageDialog("Failed to get column information.", self~hwnd, "Windows API
 Error")
  end
```

15.42. getColumnOrder

```
>>--getColumnOrder-------------------------------><
```

Retrieves the current column order of the list-view when in report view. The order of the columns can be changed programmatically by using the *setColumnOrder*() method. In XP and later there is also an *addExtendedStyle* style which lets the user rearrange the order of the columns. If the list-view control has not had the columns rearranged, then the column order will always be 0, 1, 2, ... of course.

One practical use of this method would be in conjunction with the *setColumnOrder*() method. In an application where the user can rearrange the column order, the programmer could get the order when a dialog closes, save it, and then restore the order to the user's preference when the application is restarted.

Note that the list-view control does not have to have the HEADERDRAGDROP style for the get and set column order methods to work. That style is only needed to allow the user to rearrange the columns using drag and drop. The programmer can use *setColumnOrder*() to change the order of the columns whether or not the list-view has the HEADERDRAGDROP style, or not.

Arguments:

There are no arguments.

Return value:

The possible return values are:

an **`.Array`** object

An array is returned with the index numbers of the columns. The number at the first position in the array is the column index of the left-most column. The number in the second position in the array is the index of the second left-most column, etc..

`.nil`

`.nil` is returned for some unexpected error. This is unlikely.

Example:

This example prints out the column order when the dialog is first initialized, and then prints out the order when the dialog closes.

```
  ...
  list = self~newListView(100)

  columnNames = .array~of("Occupation", "Name", "Age", "Sex")
  list~addExtendedStyle("FULLROWSELECT GRIDLINES CHECKBOXES HEADERDRAGDROP")

  list~insertColumn(0, columnNames[1], 95)
  list~insertColumn(1, columnNames[2], 95)
  list~insertColumn(2, columnNames[3], 95)
  list~insertColumn(3, columnNames[4], 35)

  self~printColumnOrder
  ...

::method close

  self~printColumnOrder
  return self~ok:super

::method printColumnOrder private
  expose list columnNames

  cols = list~getColumnOrder
  say 'cols:' cols
  say 'Column count: ' list~getColumnCount
  say 'Current order:'
  if cols~isA(.array) then do c over cols
    -- Column indexes are zero-based
    c += 1
    say " " columnNames[c]
  end

/* Output might be, (assuming the user changed the order):

cols: an Array
Column count:  4
Current order:
  Occupation
  Name
  Age
  Sex

cols: an Array
Column count:
Current order:
  Age
  Name
  Sex
  Occupation

*/
```

15.43. getColumnText

```
>>--getColumnText(--index--)---------------------><
```

Retrieves the column header text for a list-view when it is in report view.

Arguments:

The single argument is:

index [required]

The zero-based column index for which to get the text.

Return value:

Returns the text for the specified column, or the empty string on error.

15.44. getExtendedStyle

```
>>-aListControl~GetExtendedStyle-----------------><
```

The **`getExtendedStyle`** method returns the extended *styles* of a **`ListView`**. The styles are returned as a string of blank delimited style keywords. These are the same keywords as those used to set the styles.

Arguments:

This method takes no arguments.

Return value:

The returned string can contain one or more of the following style keywords. See the *addExtendedStyle* method for the style descriptions:

- BORDERSELECT
- CHECKBOXES
- DOUBLEBUFFER
- FLATSB
- FULLROWSELECT
- GRIDLINES
- HEADERDRAGDROP
- INFOTIP
- LABELTIP
- MULTIWORKAREAS
- ONECLICKACTIVATE
- REGIONAL
- SIMPLESELECT
- SUBITEMIMAGES
- TRACKSELECT
- TWOCLICKACTIVATE
- UNDERLINECOLD

- UNDERLINEHOT

Example:

The following example comes from a fictious application where the user can elect to use the track select style, or not. With this style, an item can be selected by pausing the mouse over it for a period of time specified by the hover time. The application also allows the user to increase or decrease the hover time. In this example when the methods to increment or decrement the hover time are invoked, the application first checks to see if the list-view control has the track select style. Then the change is only made if the list view control does have the style.

```


::method initDialog
  expose listView

  ...
  listView = self~newListView(IDC_LIST)
  if listView == .nil then return

  listView~addExtendedStyle("CHECKBOXES FULLROWSELECT TRACKSELECT")
  ...

::method onCheckboxClick
  expose listView

  chkBox = self~newCheckBox(IDC_CHECK_USETRACKSELECT)
  if chkBox~checked then
    listView~addExtendedStyle("TRACKSELECT")
  else
    listView~removeExtendedStyle("TRACKSELECT")

::method onDecrement
  expose listView

  styles = listView~getExtendedStyle
  if styles~wordpos("TRACKSELECT") == 0 then return

  time = listView~getHoverTime - 100
  if time > 0 then listView~setHoverTime(time)

::method onIncrement
  expose listView

  styles = listView~getExtendedStyle
  if styles~wordpos("TRACKSELECT") == 0 then return

  -- Let the user increase the hover time to any length they want.
  listView~setHoverTime(list~getHoverTime + 100)
```

15.45. getExtendedStyleRaw

```
>>-aListControl~GetExtendedStyleRaw--------------><
```

The **`getExtendedStyleRaw`** method returns the extended *styles* of a **`ListView`** as the numeric value of the extended style flags. This method is provided for programmers familiar with the Windows API who may wish to know, or to work with, the actual value of the flags.

Arguments:

This method takes no arguments.

Return value:

The returned value is the numeric value of the extended style flags currently in effect for the list-view control.

Example:

The following example is the same example as the *getExtendedStyle* example. It simply replaces the **`getExtendedStyle`** invocation with a **`getExtendedStyleRaw`** invocation.

```
::method initDialog
  expose listView

  ...
  listView = self~newListView(IDC_LIST)
  if listView == .nil then return

  listView~addExtendedStyle("CHECKBOXES FULLROWSELECT TRACKSELECT")
  ...

::method onCheckboxClick
  expose listView

  chkBox = self~newCheckBox(IDC_CHECK_USETRACKSELECT)
  if chkBox~checked then
    listView~addExtendedStyle("TRACKSELECT")
  else
    listView~removeExtendedStyle("TRACKSELECT")

::method onDecrement
  expose listView

  if .DlgUtil~and(listView~getExtendedStyleRaw, "0x00000008") == 0 then return

  time = listView~getHoverTime - 100
  if time > 0 then listView~setHoverTime(time)

::method onIncrement
  expose listView

  if .DlgUtil~and(listView~getExtendedStyleRaw, "0x00000008") == 0 then return

  -- Let the user increase the hover time to any length they want.
  listView~setHoverTime(list~getHoverTime + 100)
```

15.46. getFullRow

```
>>--getFullRow(--itemIndex--+-----------------+--)--------------><
                            +-,-useForSorting-+
```

Returns a *LvFullRow* object for the specified item in this list view.

Arguments:

The arguments are:

itemIndex [required]

The index of the list-view item to get the full row for.

sortable [optional]

The *sortable* argument must be true or false. If true, the ooDialog framework sets up some internal housekeeping that allows internal sorting of list-view items. If *sortable* is false, this internal housekeeping is skipped. The default if *sortable* is omitted is false. See the remarks section for more details.

Return value:

On success, returns a **`LvFullRow`** object whose attributes are the attributes of the list-view item specified. On error the **`.nil`** object is returned.

Remarks:

`LvFullRow` objects reflect the attributes of a list-view item and all the subitems of that item. It is common to think of a row when the list-view is in report view. However, once the subitems are added to a list-view item, they exist even when the view of the list-view is changed to one of the other views, like icon, or small icon.

The *sortItems* method is used to do custom sorting of list-view items. The sorting is dependent on the item *data* being set for each list-view item. In addition, the *sortItems* method can be used to signal the ooDialog framework to use native internal code for the sorting. For the internal sort to work, the item data item *must* be a **`LvFullRow`** object.

The returned object from *getFullRow* can be set as the item data to allow the internal sorting. When invoking the *setItemData* method with a **`LvFullRow`** object as the argument, the internal house keeping that the ooDialog framework needs to do for internal sorts is done automatically. However, if the returned full row object is instead used to insert or add a new item into the list-view, this internal house keeping may be bypassed. For this case, the *sortable* argument should be used and set to true.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example loops through all the items in a list-view, gets a full row for each item, and uses the full row as the user item data. Once this is done, the list-view items can be sorted using ooDialog's internal method. **Note** that the *sortable* argument is not used here. Invoking *setItemData* using the full row object is sufficient:

```
list = self~newListView(IDC_LV_CONTACTS)

do i = 1 to list~items
  j = i - 1
  row = list~getFullRow(j)
  list~setItemData(j, row)
end
```

This example is similar to the first example, however, the full row object is used to insert a new item into the list-view. **Note** that the *sortable* argument is used here. Once the loop finishes, the entire list can be sorted using the ooDialog internal sort.

```
list = self~newListView(IDC_LV_CONTACTS)

do i = 1 to list~items
  j = i - 1
  row = list~getFullRow(j, .true)
  list~addFullRow(row)
end
```

15.47. getHoverTime

```
>>--getHoverTime---------------------------------><
```

This method retrieves the current hover time. The hover time is expressed as milliseconds. The hover time only affects list-view controls that have the track select, one click activate, or two click activate extended list-view styles. See the *addExtendedStyle* method for a description of these styles.

Arguments:

This method takes no arguments.

Return value:

The hover time for the list-view. A value of -1 indicates the hover time is the system default hover time.

Example:

The following example comes from a program where the one click activate and track select styles are enabled. Originally the hover time is set to .4 seconds. However the application allows the users to increase or decrease this time to match their preference. A menu item (or a button could be used) is connected to the **`onIncrement`** and **`onDecrement`** methods. When either of the methods is invoked, the current hover time is examined and then either incremented or decremented by .1 of a second.

```
::method initDialog
  expose list

  ...
  list = self~newListView(IDC_LIST)
  list~addExtendedStyle("ONECLICKACTIVATE TRACKSELECT FULLROWSELECT")
  list~setHoverTime(400)
  ...

::method onDecrement
  expose list

  time = list~getHoverTime - 100
  if time > 0 then list~setHoverTime(time)

::method onIncrement
  expose list

  -- Let the user increase the hover time to any length they want.
  list~setHoverTime(list~getHoverTime + 100)
```

15.48. getImageList

```
>>--getImageList(--+--------+--)-----------------><
                   +--type--+
```

Retrieves the current image list for the type specified. The default is to retrieve the image list for the normal icons. See the *setImageList* method for information on assigning an image list to a list-view control.

Arguments:

The single argument is:

type [optional]

Specifies which image list to retrieve. The list-view maintains 4 different image lists. One for the normal icons, one for the small icons, one for the state icons, and one for the group header icons. However, ooDialog does not yet have support for list-view groups, or the group header image list.

The following keywords are used to specify which image list is to be retrieved, case is not significant:

NORMAL for the normal icon image list.	SMALL for the small icon image list.
STATE for the state image list.	

The default value if this argument is omitted is NORMAl.

Return value:

This method returns the current *ImageList*, if there is one, or **`.nil`** if there is no current image list of the type specified.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

This example comes from a complex application that opens and closes a large number of dialogs and uses many list-view controls. All the list-view controls share the same image lists. When the last dialog is closed, the shared image lists are released to free up system resources as the application enters its next phase.

```
::method ok
  self~checkImageLists
  return self~ok:super

::method cancel
  self~checkImageLists
  return self~cancel:super

::method checkImageLists
  expose list

  if self~isLastDialog then do
    types = .array~of('NORMAL', 'SMALL', 'STATE')

    do type over types
      il = list~getImageList(type)
      if il \== .nil then il~release
    end
  end
```

15.49. getItem

```
>>--getItem(--_item--)---------------------------><
```

Returns some, or all, of the information the list-view maintains on the specified item. The information is returned as a *LvItem* object. The item is specified by either a **`LvItem`** object or by the item's index.

Arguments:

The single argument is:

_item [required]

Specifies the list-view item whose information is required. *_item* can either be a *LvItem* object, or the zero-based index of the item.

When *_item* is a **LvItem** object then it can be used to only retrieve some of the list-view item's information. The value of the *mask* attribute determines which information is retrieved. The attributes of the **LvItem** specified by the mask are set to the current value of that information in the list-view's item. Attributes not specified by the mask are ignored.

When *_item* is the index of a list-view item, then a new **LvItem** object is returned. The attributes of this object reflect all the information of the list-view's item.

Return value:

When *_item* is a **LvItem** object, this method returns true on success and false on error. On success the attributes of the *_item* object, which is a **LvItem** object reflect the information requested. On error, the value of the attributes are undefined.

When *_item* is the index of a list-view item, then on success, this method returns a **LvItem** object whose attributes represent the attributes of the list-view item. On error, the **.nil** object is returned.

Remarks:

Whether to use a **LvItem** object or an item index for the *_item* argument is dependent on what the programmer sees as the most convenient. If the programmer only wishes to determine the group ID of the item and does not need any other information on the item, it may be more convenient to use a **LvItem** object with the *mask* attribute set to just *GROUPID*. On the other hand, if the programmer wants all the current information for a list-view item, it may be more convenient to specify the index of the item and get back a new **LvItem** object.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows a generic method in a dialog to get the group ID of any, arbitrary, item in a list-view:

```
::method getGroupID private
    expose list
    use strict arg itemIndex

    lvItem = .LvItem~new(itemIndex, , , , 'GROUPID')

    if list~getItem(lvItem) then do
        return lvItem~groupID
    end
    else do
        -- Some error happened, very unlikely.
        -- Handle the error
    end
```

This example shows a generic method for producing a new **LvItem** object for any item in a list-view:

```
::method getLvItem
    use strict arg listView, itemIndex

    lvItem = listView~getItem(itemIndex)
    if lvItem == .nil then do
        -- Some error happened, very unlikely.
        -- Handle the error
    end
    else do
        return lvItem
    end
```

15.50. getItemData

```
>>--getItemData(--index--)-----------------------><
```

Gets the item data associated with the specified list-view item.

Arguments:

The single argument is:

index [required]

The zero-based index of the item to retrieve the item data from.

Return value:

Returns the item data associated with the specified item, or the **`.nil`** object if there is no data associated with the item.

Remarks:

The list-view control allows the user (the Rexx programmer) to associate a data value with any, or all, of the list-view items. The data value can be any ooRexx object. The data value is set using the *setItemData* method. If needed the data value can be removed from a list-view item through the *removeItemData* method. Storing a user value for each item can be useful in any number of ways. One specific use is in the *sortItems* method.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example uses the data value stored for each item to display extra employee information when the user double clicks on an item in a list-view that displays each employee. Note that the Rexx object stored for each list-view item is a **`Directory`** object:

```
::method onDoubleClick
  expose list
  use arg id, index, subitem, flags

  tab = '9'x

  if index <> -1 then do
    data = list~getItemData(index)

    msg = 'Extra Information for' data~fName data~lName || .endOfLine~copies(2)

    select
      when subItem == 0 then do
        msg ||= 'Employee ID:' || tab           || data~ID      || .endOfLine
        msg ||= 'Manager:'     || tab~copies(2) || data~manager || .endOfLine
      end
      when subitem == 1 then do
        msg ||= 'Telephone:' || tab || data~telephone || .endOfLine
        msg ||= 'Cellphone:' || tab || data~cell      || .endOfLine
      end
      when subitem == 2 then do
        msg ||= 'Pay scale:' || tab || data~scale     || .endOfLine
        msg ||= 'Status:'    || tab || data~empStatus || .endOfLine
      end
      otherwise do
        -- insurance, can't happen
        nop
      end
    end
    -- End select

    title = 'Acme Plumbing Employee Statistics'

    z = MessageDialog(msg, self~hwnd, title, 'OK')
  end

  return 0
```

15.51. getItemInfo

```
>>--getItemInfo(--index--,--directory--+------------+--)--------><
                                       +-,-subItem--+
```

Returns the attributes of an item, or one of its subitems, in a **`Directory`** object.

Arguments:

The arguments are:

index

The zero-based index of the item whose information is to be retrieved. If the *subItem* argument is specified, then the information for the subitem of this item is returned.

directory [required]

A **`Directory`** object whose indexes will be filled with the returned item information. On success, the **`Directory`** object will have the following indexes with the corresponding information:

text

The text of the item, or subitem.

image

The index in the image list for the icon, (both big and small,) for the item. If the item does not have an icon in image list, the value will be -1.

state

Zero or more blank separated keywords indicating the state of the item. The possible state keywords are:

CUT

The item is marked for a cut-and-paste operation.

DROP

The item is highlighted as a drag-and-drop target.

FOCUSED

The item has the focus, so it is surrounded by a standard focus rectangle. Although more than one item may be selected, only one item can have the focus.

SELECTED

The item is selected. The appearance of a selected item depends on whether it has the focus and also on the system colors used for selection.

subitem [optional]

The one-based index of a subitem. If *subitem* is omitted, or zero, the information for the item with the *index* specified is retrieved. If a subitem index is specified, the information for the item's specified subitem is retrieved instead.

Return value:

Returns **.true** on success and **.false** on error.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

This example displays information on a selected item in the list-view:

```
::method displayItemInfo

  listView = self~newListView(IDC_LV_JOBS_PER_STATE)

  title = "Selected Item Information"
  buttons = "ok"
  icon = "information"

  item = listView~selected
  if item == -1 then do
    msg = "No item is selected."
    buttons = "cancelTryContinue"
  end
  else do
    d = .Directory~new
    if listView~getItemInfo(item, d) then do
      msg = "Item Text  : "d~text   || .endOfLine || -
            "Image Index: "d~image  || .endOfLine || -
            "Item State : "d~state
    end
    else do
      msg = "Windows API error. No information available."
      buttons = "abortRetryIgnore"
      icon = "error"
    end
  end

  return MessageDialog(msg, self~hwnd, title, buttons, icon)
```

15.52. getItemPos

```
>>--getItemPos(--index--)------------------------><
```

Retrieves the *client area* coordinates of an item in a list-view control as a **Point** object.

Arguments:

The single arguments is:

index

The zero-based index of the item whose position is needed.

Return value:

On success, returns a *Point* object containing the position of the upper left corner of the item specified, relative to the client area of the list-view.

On error, 0 is returned. See the remarks

Remarks:

Since quite often list-view controls contain more items than can fit in the client area of the list-view, the position returned by *getItemPos* may be outside of the client area rectangle. This implies that both the x and y coordinates in the returned position may be negative. For instance, take a list-view that has 400 items and more columns than fit in the list-view. If the user has scrolled the list-view all the way to the bottom right and *getItemPos* is used to get the position of item 5, both the x and y position will be negative.

Experimentation has shown that if the *index* specified is larger than the actual number items in the list-view, the list-view still returns a position. The position seems to be correct for the *index*, if the list-view did have that item. Because of this, it seems unlikely this method will ever fail.

15.53. getItemRect

```
>>--getItemRect(--index--+---------+--)----------><
                         +-,-part--+
```

Gets the bounding *rectangle* for all or part of an item in the current view.

Arguments:

The arguments are:

index [required]

The index of the list-view item that contains the subitem whose bounding rectangle is to be retrieved.

part [optional]

Keyword indicating which part of the item rectangle is sought. Use exactly one of the following keywords, case is not significant:

BOUNDS	LABEL
ICON	SELECTBOUNDS

BOUNDS

Returns the bounding rectangle of the entire item, including the icon and label. This is the default if this argument is omitted.

ICON

Returns the bounding rectangle of the icon or small icon.

LABEL

Returns the bounding rectangle of the item text.

SELECTBOUNDS

Returns the union of the ICON and LABEL rectangles, but excludes columns in report view.

Return value:

The specified bounding rectangle, as a *Rect* object, on success, the **`nil`** object on error.

Remarks:

Note that a bounding *rectangle* is different than a *point / size* rectangle.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example is a snippet of code that overlays a combo box on top of a list-view. The combo box is sized to occupy the area of the label of the list-view item at *itemIndex*. Note that the combo box's height is set to 4 times the hit of the list-view item. When the combo box is first shown, it does not show the drop-down list and its height is very close to the height of the list-view item. However, its height is restricted to the height of the item, the drop-down list will not show when the user clicks on the down arrow icon. Setting its height larger allows the drop-down list to open and show 3 items:

```
 ...

 r = list~getItemRect(itemIndex, 'LABEL')
 ret = cb~setWindowPos(list~hwnd, r~left, r~top, r~right - r~left, 4 * (r~bottom -
r~top), "SHOWWINDOW NOZORDERCHANGE")
 cbVisible = .true
 cb~assignFocus
```

15.54. getSubitem

```
>>--getSubitem(--_item--+----------------+--)----><
                        +-,-subItemIndex-+
```

Receives some, or all, of the information of a subitem of a list-view item. The information is returned as a *LvSubItem* object. The subitem is specified by either a **`LvSubItem`** object, or by a item index / subitem index combination.

Arguments:

The arguments are:

_item [required]

Specifies the item whose subitem information is to be retrieved.

If *_item* is a **`LvSubItem`** object, then the *item* and *subItem* attributes of the **`LvSubItem`** object specify which subitem to query. The *mask* attribute specifies which information is desired.

If the *_item* argument is not a **`LvSubItem`** object, then it must be the zero-based index of the item to query and the *subItemIndex* argument must be the one-based index of the subitem.

subitemIndex [optional]

Specifies the index of the subitem. If *_item* is a **`LvSubItem`** object then the *subitemIndex* is ignored. If *_item* is not a **`LvSubItem`** object then *subitemIndex* is required and specifies the one-based index of the subitem to be queried.

Return value:

When the *_item* argument is a **`LvSubItem`** object then this method returns true on success and false on error. On success, the attributes of *_item* reflect the information queried of the subitem. On error, the value of the attributes of *_item* are undefined.

When the *_item* argument is not a **`LvSubItem`** object then, on success, this method returns a **`LvItem`** object whose attributes reflect the attributes of the list-view subitem. On error, the **`.nil`** object is returned.

Remarks

The choice of whether to use a **`LvItem`** object or an item index / subitem index combination to specify which subitem to query is up to the programmer. It is dependent on which seems the most convenient to the programmer for the task.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example queries subitem 2 of the item at index 8 of the list-view for just its image index. On return, the *imageIndex* attribute of the *LvSubItem* object will reflect the image index of the specified subitem. The value of the *text* attribute is undefined:

```
lvSubItem = .LvSubItem~new(8, 2, , , 'IMAGE')
if list~getSubItem(lvSubItem) then do
    -- Do something with the information.
    ...
end
```

This example queries the same subitem by using the item / subitem indexes. On success a *LvSubItem* object is returned that reflects all the information for the subitem.

```
lvSubItem = list~getSubItem(8, 2)
if lvSubItem \== .nil then do
    -- Do something with the LvSubItem object.
    ...
end
```

15.55. getSubitemRect

```
>>--getSubitemRect(--index-,-subitemIdx--+---------+--)---------><
                                         +-,-part--+
```

Gets the bounding *rectangle* for all or part of a subitem in the current view of this list-view control.

Arguments:

The arguments are:

index [required]

The index of the list-view item that contains the subitem whose bounding rectangle is to be retrieved.

subitemIdx [required]

The index of the list-view subitem that contains the subitem whose bounding rectangle is to be retrieved. This index, may be 0, in which case, the *getSubitemRect* method works exactly the same as the *getItemRect*, except that the SELECTBOUNDS keyword is not accepted.

part [optional]

Keyword indicating which part of the item rectangle is sought. Use exactly one of the following keywords, case is not significant:

BOUNDS	ICON	LABEL

BOUNDS

Returns the bounding rectangle of the entire subitem, including the icon and label. This is the default if this argument is omitted.

ICON

Returns the bounding rectangle of the icon or small icon of the subitem.

LABEL

Returns the bounding rectangle of the subitem text.

Return value:

The specified bounding rectangle, as a *Rect* object, on success, the **`nil`** object on error.

Remarks:

Note that a bounding *rectangle* is different than a *point / size* rectangle.

Details

Requires Windows Vista or later. Or, if the Rexx program is running on an earlier version of Windows, the list-view **must** be in report view.

Raises syntax errors when incorrect usage is detected.

Example:

This example is a snippet of code that overlays a combo box on top of a list-view. The combo box is sized to occupy the area of the label of the list-view subitem at the specified item index and subitem index. Note that the combo box's height is set to 4 times the hit of the list-view subitem. When the combo box is first shown, it does not show the drop-down list and its height is very close to the height of the list-view subitem. However, its height is restricted to the height of the subitem, the drop-down list will not show when the user clicks on the down arrow icon. Setting its height larger than the subitem allows the drop-down list to open and show 3 items:

```
  ...

  if columnIndex > 0 then do
      r = list~getSubitemRect(itemIndex, columnIndex, 'LABEL')
      ret = cb~setWindowPos(list~hwnd, r~left, r~top, r~right - r~left, 4 * (r~bottom -
r~top), "SHOWWINDOW NOZORDERCHANGE")
      cbVisible = .true
      cb~assignFocus
  end
```

15.56. getToolTips

```
>>--getToolTips----------------------------------><
```

Retrieves the child *ToolTip* control used by this list-view.

Arguments:

This method has no arguments.

Return value:

Returns the child Rexx **`ToolTip`** object, or the **`.nil`** object if the list-view does not have a child tool tip control.

15.57. getView

```
>>--getView--------------------------------------><
```

Retrieves the current view of this list-view.

Arguments:

There are no arguments for this method.

Return value:

Returns a keyword indicating the current view of this list-view.

ICON	LIST
SMALLICON	REPORT

ICON

The list-view is in icon view. Each item appears as a full-sized icon with a label below it.

SMALLICON

The list-view is in small icon view. Each item appears as a small icon with the label to the right of it.

LIST

The list-view is in list view. Each item appears as a small icon with a label to the right of it. Items are arranged in columns.

REPORT

The list-view is in report view. Each item appears on its own line, with information arranged in columns.

Remarks:

Use the *setView* method to change the current view in the list-view.

Details

Requires Common Control *Library* version 6.0 or later.

Raises syntax errors when incorrect usage is detected.

15.58. hasCheckBoxes

```
>>--hasCheckBoxes--------------------------------><
```

Determines if the list-view has the CHECKBOXES extended style.

Arguments:

This method has no arguments.

Return value:

Returns **`.true`** if the list-view has the CHECKBOXES extended style, otherwise **`.false`** .

15.59. hitTestInfo

```
Form 1:

>>--hitTestInfo(--pos--+---------+--)------------><
                       +-,-info--+

Form 2:

>>--hitTestInfo(--x--,--y--+---------+--)--------><
                           +-,-info--+

Generic Form:

>>--hitTestInfo(--hitPoint--+---------+--)-------><
                            +-,-info--+
```

Determines which list-view item, if any, is at the specified position.

Arguments:

The arguments are:

hitPoint [required]

The point, in client *coordinates*, to test.

info [optional in / out]

A **`.Directory`** object in which additional information concerning the hit test will be returned. This argument can be omitted if the additional information is not needed. When the argument is used, on return *info* will contain some, or all, of the following indexes, depending on the version of Windows:

INFO

A string of keywords containing information about the hit test. One or more of the following keywords will be present:

Above	ToLeft	OnLabel
Below	Nowhere	OnStateIcon
ToRight	OnIcon	OnItem

Above

The position is above the list-view's client area.

Below

The position is below the list-view's client area.

ToRight

The position is to the right of the list-view's client area.

ToLeft
: The position is to the left of the list-view's client area.

Nowhere
: The position is inside the list-view control's client window, but it is not over a list item.

OnIcon
: The position is over a list-view item's icon.

OnLabel
: The position is over a list-view item's text.

OnStateIcon
: The position is over the state image of a list-view item.

OnItem
: The position is over a list-view item. This is a combination of *OnIcon*, or *OnLabel*, or *OnStateIcon*.

INFOEX

This index will only be present on Vista or later and is a string of keywords containing extended information about the hit test. The index may contain one or more of the following keywords:

Footer	GroupCollapse	GroupStateIcon
Group	GroupFooter	GroupSubsetLink
GroupBackground	GroupHeader	OnContents

Footer
: The point is within the footer of the list-view control.

Group
: The point is within a group. This is a combination of *GroupBackground*, or *GroupCollapse*, or *GroupFooter*, or *GroupHeader*, or *GroupStateIcon*, or *GroupSubsetLink*.

GroupBackground
: The point is within the area of the group where items are displayed.

GroupCollapse
: The point is within the collapse/expand button of the group.

GroupFooter
: The point is within the group footer.

GroupHeader
: The point is within the group header.

GroupStateIcon
: The point is within the state icon of the group.

GroupSubsetLink
: The point is within the subset link of the group.

OnContents
: The point is within the icon or text content of the item and not on the background.

ITEM
: The index of the item if the point hits an item or subitem. -1 if the point did not hit an item or subitem.

SUBITEM
: The subitem index if the point hits a subitem, otherwise 0.

GROUP
: Vista or later only. The group index of the item hit. This is valid only for owner data. If the point is within an item that is displayed in multiple groups then GROUP will specify the group index of the item.

Return value:

The index of the list-view item hit, or -1 if the point does not hit an item.

Remarks:

Note that the values in the **`Directory`** object reflect what the operating system returns. Experimentation shows that the values returned can vary depending on the version of Windows a program is executing on. Some values also differ depending on the style(s) of the list-view. For instance, the presence or absence of the *extended* FULLROWSELECT style changes some of the values when hit testing the exact same spot. In addition, Microsoft documents some values, such as ABOVE, that are never seen during testing.

Sometimes the returned index of the hit spot is all that is needed. In these cases, there is no need to supply the optional **`.Directory`** object. However, other times the complete hit test information may be desired.

Any x, y coordinates will work. I.e. -6000, -7000 will work. The item index will be -1 and location will be *Above ToLeft*.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

Example:

This example is from a program that allows the user to edit the text of subitems. The code snippet checks if the point p is over an item's subitem and if so invokes the editing method:

```
listView = self~newListView(IDC_LV_PLAYERS)

d = .Directory~new
index = listView~hitTestInfo(p, d)
if index <> -1 & d~subItem <> 0 then do
  self~subItemEdit(index, d~subItem)
  ...
end
```

15.60. insert

```
>>--insert(--+------+--,--+---------+--,--text--+--------+--)---><
             +-item-+     +-subitem-+           +-,-icon-+
```

The **`insert`** method inserts a new item into the list-view control, or inserts the text of a subitem.

Remember that in a list-view, items and columns use a 0-based index. Subitems use a 1-based index.

Arguments:

The arguments are:

item

The index of the item to insert or the item whose subitem text is inserted.

When this argument is omitted **and** *subitem* is omitted or 0, ooDialog uses the index of the last item added to the list-view control, plus 1, as the item index. If the argument is omitted and *subitem* is greater than 0, then the index of the last item added is used. If no items have yet been added, then 0 is used.

subitem

The subitem number. This argument is used to insert the text for a subitem of the item. In report view, the label (text) for the item is placed in column 0. The text for subitem 1 is placed in column 1, the text for subitem 2 is placed in column 2, and so on. If you omit this argument, or use 0, the *text* argument is assigned as the label (text) of the item. If you use this argument and it is greater than 0, then the *text* argument is assigned as the text of that subitem. **Note** that when this argument is greater than 0, no new item is inserted into the list-view. Rather, the text of the subitem is inserted into an existing list-view item.

text

The text of the item, or of the subitem, depending on the *subitem* argument.

icon

The index of the icon within the image list for this item. Use the *setImageList*() method to assign an image list to the list-view control. With **`setImageList`**(), use the LVSIL_SMALL type to set the image list for the small icons and the LVSIL_NORMAL type for the normal size icon image list.

The normal size icons are used for the icon view and the small icons are used for the list, report, and small-icon views. Note that there is only one index for each list-view item. This implies that the normal icon for an item must be at the same index in the normal image list as the small icon is at in the small icon image list.

Use -1 or simply omit this argument to indicate there is no icon for the item being inserted. This argument is ignored completely if *subitem* is greater than 0.

Return value:

The index of the new item, or -1 in all other cases.

Example:

The following example inserts items in a list:

```
::method initReport
  curList = self~newListView(102)
  if curList \= .nil then
  do
    curList~insert(, ,"First")
    curList~insert(, ,"Second")
    curList~insert(, ,"Third")
  end
```

15.61. insertColumn

```
>>--insertColumn(--+----------+-txt-,-width-+-------+-+------------------+--)--><
                   +--colIdx--+             +-,-fmt-+ +-,-adjustFullRows-+
```

Inserts a new column into this list-view, using *dialog* units to specify the width.

Arguments:

The arguments are:

colIdx [optional]

The zero-based index of the column to insert. The default if this argument is omitted is 0.

txt [required]

The text, the label, for the column. The text for the column must be less than 256 characters long.

width [required]

The width, in dialog units, for the inserted column.

The width value will usually produce *inaccurate* results because it is converted to pixels using *factorX*. The operating system requires the width be specified in pixels. The programmer is advised to use the *insertColumnPX* method instead, which will correctly assign the width.

ftn [optional]

A keyword specifying the format for the column. Use exactly one of the following keywords:

CENTER	LEFT	RIGHT

CENTER

Text of the column is center aligned

LEFT

Text of the column is left aligned. This is the default if the *fmt* argument is omitted.

RIGHT

Text of the column is right aligned.

adjustFullRows [optional]

If *adjustFullRows* is true, the ooDialog framework attempts to fix up the *LvFullRow* objects assigned to the item *data* of each list-view item. Nothing is done if the arguemnt is false. The default is false. This argument is only used to support *internal* sorting. If the application is not using internal sorting then this argument should just be ignored.

Return value:

Returns the column index of the inserted column on success, -1 on error.

Remarks:

If *colIdx* is greater than the current number of columns then the method does not fail. Rather, the operating system adds the column as the last column rather that at the index specified. I.e., if there currently are 2 columns, (column 0 and column 1,) and *colIndex* is 3, the operating system adds the column at index 2 rather than index 3.

Only use the *adjustFullRows* argument to adjust the **`LvFullRow`** objects if every list-view item has a **`LvFullRow`** item assigned as the item data, and the programmer has kept the full rows consistent with changes to the column count of this list-view. If these conditions are not met, this method will return an error code, even if the column was inserted successfully. However, no harm is done, it will just make it hard to tell between a successful execution of the method and an error.

Details

Raises syntax errors when incorrect usage is detected.

15.62. insertColumnPX

```
>>--insertColumnPX(-+--------+-,-txt-,-width-+-------+-+-----------------+-)--><
                    +-colIdx-+               +-,-fmt-+ +-,-adjustFullRows-+
```

Inserts a new column into the list-view specifying the column width in pixels.

Arguments:

The arguments are:

colIdx [optional]

The zero-based index for the column being inserted. Columns are numbered from the left to the right. The default if this argument is omitted is column 0.

txt [required]

The text (label) for the column. The text for the column must be less than 256 characters long.

width [required]

The width of the column being inserted, specified in pixels.

fmt [optional]

One of the following keywords, case is not significant. The keyword specifies the format for the column being inserted. At this time not all formats for the column are recognized. Other format keywords may be added in the future.

LEFT	CENTER	RIGHT

LEFT

Text in the column is aligned to the left. This is the default if this argument is omitted.

CENTER

The text for the column is centered.

RIGHT

The text for the column is aligned right.

adjustFullRows [optional]

If *adjustFullRows* is true, the ooDialog framework attempts to fix up the *LvFullRow* objects assigned to the item *data* of each list-view item. Nothing is done if the arguemnt is false. The default is false. This argument is only used to support *internal* sorting. If the application is not using internal sorting then this argument should just be ignored.

Return value:

Returns the index of the newly inserted column on success, otherwise -1.

Remarks:

The inserted columns are only visible when the list-view is in report view, but, columns can be inserted into the list-view control when it is in any view.

This method and the *insertColumn* method are virtually the same, except that in this method the width of the column is specified in pixels. The *insertColumn* method takes dialog units for the width and then converts them to pixels using an *inaccurate* method. This makes the width used by the *insertColumn* method inaccurate.

Note that previous versions of this documentation incorrectly stated that the *insertColumn* used pixels to specify the width.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

This example inserts 3 columns into the list-view control. Each column will be 50 pixels wide and left aligned.

```
::method initReportView
  lv = self~newListView(IDC_LV_STATS)

  lv~insertColumnPX(0, "First Name", 50)
  lv~insertColumnPX(1, "Last Name", 50)
  lv~insertColumnPX(2, "Age", 50)
```

15.63. insertFullRow

```
>>--insertFullRow(--row--)-----------------------><
```

Adds an item to the list-view by inserting it into the list at the position specified by the *row* argument.

Arguments:

The single argument is:

row [required]

A *LvFullRow* object that specifies the item to be added to the list-view.

Return value:

On success, returns the zero-based index of the added row. On error, -1 is returned.

Remarks:

When a full row object is used to add an item to the list view, all the subitems are also added. The *insertFullRow* method always inserts the item into the list at the position specified in the full row object. The item index in the subitems in the full row object are ignored. After the item is inserted, the item indexes in the subitems are updated to the correct value of the newly added item.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example constructs a full row object and inserts it into the list-view. The row is inserted at index 120 of the list-view:

```
  lvItem = .LvItem~new(120, "Business manager", 6)
  lvSub1 = .LvSubItem~new(120, 1, "Tom", 14)
  lvSub2  = .LvSubItem~new(120, 2, "Sawyer", 26)
  lvSub3  = .LvSubItem~new(120, 3, "ts@google.com", 11)
  lvFullRow = .LvFullRow~new(lvItem, lvSub1, lvSub2, lvSub3, .true)
  list~insertFullRow(lvFullRow)
```

15.64. isChecked

```
>>-aListControl~isChecked(--index--)-------------><
```

The **`isChecked`** method determines if a check box for a list-view item is checked or not. This method will always return true or false. It can be used if the list-view control has or does not have the check boxes extended style. (See the *addExtendedStyle* method for a description of the check boxes extended style.)

Arguments:

The only argument is:

index

The index of the item to query.

Return value:

The return values are:

true

The list-view item has a check box and it is checked.

false

There is no check mark for the list-view item. Either the list view has check boxes and the check box is not checked, the index is out of range or not valid, or the list-view does not have the check box extended style.

Example:

The following example is from a doctor's office custom accounting application. A list box is filled with information for every patient. After the office manager has finished working with the accounts a mail merge is done to produce statements for all patients that are to receive statements for the month. For every patient item in the list-view, if the item has a check mark, then that patient is mailed a statement.

```


::method onCreateMailMerge
  expose patientList

  do i = 0 to patientList~items
    if billingList~isChecked(i) then do
      self~addToMailMerge(billingList, i)
    end
  end
```

15.65. itemInfo

```
>>--itemInfo(--item--,--+--------+--)------------><
                        +-column-+
```

The itemInfo method retrieves the attributes of a list-view item as a **`Stem`** object.

Arguments:

The arguments are:

item

The number of the item.

column
: The number of the column. If you omit this argument, 0 is assumed.

Return value:

A compound variable that stores the attributes of the item, or -1 in all other cases. The compound variable can be:

RetStem.!TEXT
: The item text.

RetStem.!IMAGE
: The index of the icon in the image list this for item. Image lists are assigned to list-views using the *setImageList*() method.

  In report view, this index is only used for the icon for the first column. ooDialog does not yet support setting the icon index for the other columns in a report view.

RetStem.!STATE
: One or more of the following values:

  CUT
  : The item is marked for a cut-and-paste operation.

  DROPPED
  : The item is highlighted as a drag-and-drop target.

  FOCUSED
  : The item has the focus and is therefore surrounded by the standard focus rectangle. Only one item can have the focus.

  SELECTED
  : The item is selected. Its appearance depends on whether it has the focus and on the system colors used for a selection.

15.66. itemPos

```
>>-aListControl~itemPos(--item--)----------------><
```

The itemPos method retrieves the position of the upper left corner of the item.

Arguments:

The only argument is:

item
: The number of the item.

Return value:

The x- and y-coordinates of the upper left corner of the item, or -1 if you did not specify *item*, or 0 in all other cases.

Note

Use *defListDragHandler* to support default dragging:

```
self~connectListViewEvent(104,"BEGINDRAG","mthDefListDragHandler")
```

15.67. items

```
>>-aListControl~Items----------------------------><
```

The items method retrieves the number of items in a list-view control.

Return value:

The number of items.

15.68. itemsPerPage

```
>>-aListControl~ItemsPerPage---------------------><
```

The itemsPerPage method calculates the number of items that vertically fit the visible area of a list-view control that is in list or report view. Only fully visible items are counted.

Return value:

The number of fully visible items. If the current view is an icon or small-icon view, the return value is the total number of items in the list view control.

15.69. itemState

```
>>-aListControl~itemState(--item--)--------------><
```

The itemState method retrieves the state of a list view item.

Arguments:

The only argument is:

item

The number of the item.

Return value:

The state of the item, which can be one or more of the following values:

CUT

The item can be used in a cut-and-paste operation.

DROPPED
: The item is highlighted as a drag-and-drop target.

FOCUSED
: The item has the focus and is therefore surrounded by the standard focus rectangle. Only one item can have the focus.

SELECTED
: The item is selected. Its appearance depends on whether it has the focus and on the system colors used for a selection.

15.70. itemText

```
                                     +-0------+
>>-aListControl~itemText(--item--,--+--------+--)---------------><
                                     +-column-+
```

The itemText method retrieves the text of a list view item or a column.

Arguments:

The arguments are:

item
: The number of the item.

column
: The number of the column. If you omit this argument, 0 is assumed.

Return value:

The item or column text, or -1 if you did not specify *item*.

15.71. last

```
>>-aListControl~Last-----------------------------><
```

The last method retrieves the number of the last item in a list-view control.

Return value:

The number of the last item.

15.72. lastSelected

```
>>-aListControl~LastSelected---------------------><
```

The lastSelected method returns the number of the item selected last.

Return value:

The number of the item selected last, or -1 in all other cases.

15.73. modify

```
>>--modify(--+------+--,--+---------+--,--text--+--------+--)---><
             +-item-+     +-subitem-+           +-,-icon-+
```

The *modify* method modifies or sets the text for the item label, or the text for a subitem, for a list-view item. Optionally, it sets or modifies the icon index for a list-view item.

Remember that in a list-view, items and columns use a 0-based index. Subitems use a 1-based index.

Arguments:

The arguments are:

item [optional]
: The index of the item to be modified. If this argument is omitted, the selected item is used.

subitem [optional]
: This method works in a similar manner as the *insert*() method. The use of the *subitem* argument controls whether the *text* argument applies to the label of the item or the text of a subitem. When *subitem* is omitted or 0, then the label of the item is set to *text*. When *subitem* is greater than 0, then the text of that subitem is set to *text*.

text [required]
: The new text for the item label, or the new text for the subitem of the item, depending on the value of *subitem*.

icon [optional]
: The new index of the icon for the item within the image list assigned to the list-view. Image lists are assigned with the *setImageList*() method. When assigning the image list with **setImageList**(), use the LVSIL_NORMAL flag for the icon view icons and the LVSIL_SMALL flag for the list, report, and small-icon view icons.

  Use -1 or simply omit this argument to leave the icon index unchanged. If *subitem* is greater than 0, this argument is ignored.

Return value:

Returns 0 on success, 1 for all errors.

Example:

The following example modifies the icon for an item when it is activated. A method, **onActivate**, is connected to the ACTIVATE notification. When an item is activated it also gains the focus. The **onActivate** method is invoked when an item is activated, and the example determines which item was activated by seeing which item has the focus:

```


::method initDialog
  expose list

  list  = self~newListView(IDC_LV_VIEWS)
  self~connectListViewEvent(IDC_LV_VIEWS, 'ACTIVATE', 'onActivate')

  ...

::method onActivate
  expose list
```

```
    focusedItem = list~focused
    text = list~itemText(focusedItem)

    list~modify(focusedItem, , text, 2)
```

15.74. modifyColumn

```
>>-aListControl~modifyColumn(--column--,--+------+--,--width---->
                                          +-text-+

>--+---------------------+--)----------------------------------><
   |       +-LEFT---+    |
   +-,--"--+-CENTER-+--"-+
           +-RIGHT--+
```

The modifyColumn method sets new attributes for a column of a list-view control.

Arguments:

The arguments are:

column
: The number of the column. 0 is the first column.

text
: The text of the column heading. If you omit this argument, the heading is not changed.

width
: The width of the column, in pixels.

align
: The alignment of the column heading and the subitem text within the column. It can be one of the following values:

  CENTER
  : The text is centered.

  LEFT
  : The text is left-aligned, which is the default.

  RIGHT
  : The text is right-aligned.

Return value:

0
: The column was modified.

-1
: You did not specify *column*.

1
: For all other cases.

15.75. modifyColumnPX

```
>>--modifyColumnPX(--index--+---------+--+---------+--+--------+--)----------><
```

```
                          +-,-text--+  +-,-width--+  +-,-fmt--+
```

Modifies some, or all, of the attributes of an existing column in a list-view control.

Arguments:

The arguments are:

index [required]

The zero-based index for the column being modified. Columns are numbered from the left to the right.

text [optional]

If this argument is included, the text for the column is changed to the text specified.

width [optional]

The new width of the column, in pixels. If the argument is omitted, the width is left unchanged.

fmt [optional]

One of the following keywords separated by spaces, case is not significant. The keyword specifies the new format for the column being modified. If the *fmt* argument is omitted, the alignment is left unchanged. At this time not all formats for the column are recognized. Other format keywords may be added in the future.

LEFT	CENTER	RIGHT

LEFT

Text in the column is aligned to the left.

CENTER

The text for the column is centered.

RIGHT

The text for the column is aligned right.

Return value:

0 is returned on success, 1 on error.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

This example changes the title, size, and alignment of the first three columns in a list-view control. The columns are only visible when the list-view is in report view.

```
::method changeColumn
  lv = self~newListView(IDC_LV_AVAILABLE_AUTOS)
  lv~modifyColumn(0, "Make", 100, "RIGHT")
  lv~modifyColumn(1, "Model", 150, "RIGHT")
  lv~modifyColumn(1, "Color", 80, "CENTER")
```

15.76. modifyFullRow

```
>>--modifyFullRow(--row--)-----------------------><
```

Modifies the information for an item, and all its subitems, in a list-view using a *LvFullRow* object.

Arguments:

The single argument is:

row [required]

The *row* argument must either be a **`LvFullRow`** object, or the zero-based index of the list-view item to be modified.

LvFullRow object:

When the *row* argument is a **`LvFullRow`** object, the information in the list-view item is modified to match the information in the full row object.

Item index:

When the *row* argument is the index of the item to modify, the item *data* of the list-view item *must* have been assigned a **`LvFullRow`** object. The list-view item's information is then modified to match the information in the assigned full row object.

Return value:

Returns true on success, false on error.

Remarks:

Using an item index for the *row* argument can be thought of as a *sync* operation. It is assumed that the programmer has changed the value of the attributes of the **`LvFullRow`** object that is assigned as the item data of the list-view item. Then, invoking the *modifyFullRow* method has the effect of syncing up the underlying list-view item's information with the full row object's information.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example modifies the information for the list-view item with index 3. The item has 4 subitems and uses subitem images. The application only wants to modify the text and image index for the item and the first subitem. The other information of the item is left alone. The first step of getting the full row object for item 3, ensures that all the information in the full row object matches the current state of the item. Then just the label and icon index for the item and first subitem is changed, and the full row object is used to modify the list-view item:

```
lvfull = list~getFullRow(3)

lvFull~item~text = 'Nurse Practitioner'
lvFull~item~imageIndex = 4
lvFull~subitem(1)~text = 'Harry'
lvFull~subitem(1)~imageIndex = 14

list~modifyFullRow(lvFull)
```

15.77. modifyItem

```
>>--modifyItem(--lvItem--)-----------------------><
```

Modifies some or all of the data this list-view maintains for an item using a *LvItem* object.

Arguments:

The single argument is:

lvItem [required]

A **LvItem** object that specifies which item to modify, what data to modify, and the new data for the item.

Return value:

Returns true on success and false on error.

Remarks:

The *index* attribute of the *lvItem* object specifies which item to modify. The *mask* and *itemStateMask* attributes specify which attributes are valid, and the value of those attributes specifies the new data for the list-view item.

For instance to modify the group ID and image index only of the list-view item at index 17, the programmer would use a **LvItem** object and set the *index* attribute to 17, set the *mask* attribute to GROUPID IMAGE, set the GroupID attribute to the new value for the group ID, and set the *imageIndex* to the new index for the image.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows the process outlined in the remarks section for changing some information for the list-view item at index 17:

```
lvItem = .LvItem~new(17)
lvItem~imageIndex = 8
lvItem~groupID    = 101
lvItem~mask       = "GROUPID IMAGE"

list = self~newListView(IDC_LV_CLIENTS)

list~modifyItem(lvItem)
```

Note that the **LvItem** class will automatically set the *mask* attribute correctly as the attribute values are added. So, the above code could have simply been:

```
lvItem = .LvItem~new(17)
lvItem~imageIndex = 8
lvItem~groupID    = 101

list = self~newListView(IDC_LV_CLIENTS)

list~modifyItem(lvItem)
```

The *mask* attribute was set explicitly in the first example for clarity, to show that only the image index and group ID would be modified.

15.78. modifySubitem

```
>>--modifySubitem(--lvSubitem--)-----------------><
```

Modifies some or all of the data this list-view maintains for a subitem of an item using a *LvSubItem* object.

Arguments:

The single argument is:

lvSubitem [required]

A **LvSubItem** object that specifies which subitem to modify, what data to modify, and the new data for the subitem.

Return value:

Returns true on success and false on error.

Remarks:

To modify a subitem, set the *item* and *subItem* attributes of a **LvSubItem** object to identify which subitem is to be modified. Then set the *mask* attribute to specify what values of the subitem are to be changed, and set either or both of the *text* and *imageIndex* attributes to the new information.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example updates the image index of the second subitem in the list-view item with index 17.

```
lvSubitem = .LvSubItem~new(17, 2, , 16, 'IMAGE')

list = self~newListView(IDC_LV_CLIENTS)

list~modifySubitem(lvSubitem)
```

Note that the *Section 15.125.2, “new (Class Method)”* method of the **LvSubItem** will automatically set the *mask* attribute correctly. So, the above code could have simply been:

```
lvSubitem = .LvSubItem~new(17, 2, , 16)

list = self~newListView(IDC_LV_CLIENTS)

list~modifySubitem(lvSubitem)
```

The *mask* argument was used in the first example for clarity, to show that only the image index would be modified.

15.79. next

```
>>-aListControl~next(--item--)--------------------><
```

The next method retrieves the item that follows, or is to the right of, *item*.

Arguments:

The only argument is:

item

The number of the item at which the search is to start.

Return value:

The number of the following item, or -1 in all other cases.

15.80. nextLeft

```
>>-aListControl~nextLeft(--item--)---------------><
```

The nextLeft method retrieves the item left to *item*.

Arguments:

The only argument is:

item

The number of the item at which the search is to start. The specified item itself is excluded from the search.

Return value:

The number of the next item to the left, or -1 in all other cases.

15.81. nextRight

```
>>-aListControl~nextRight(--item--)--------------><
```

The nextRight method retrieves the item right to *item*.

Arguments:

The only argument is:

item

The number of the item at which the search is to start. The specified item itself is excluded from the search.

Return value:

The number of the next item to the right, or -1 in all other cases.

15.82. nextSelected

```
>>-aListControl~nextSelected(--item--)-----------><
```

The nextSelected method retrieves the selected item that follows, or is to the right of, *item*.

Arguments:

The only argument is:

item

The number of the item at which the search is to start. The specified item itself is excluded from the search.

Return value:

The number of the selected item, or -1 in all other cases.

15.83. prepare4nItems

```
>>-aListControl~prepare4nItems(--items--)--------><
```

The prepare4nItems method prepares a list-view control for adding a large number of items.

Arguments:

The only argument is:

items

The number of the items to be added later.

Return value:

0

The list-view control was prepared.

-1

You did not specify *items*.

15.84. prependFullRow

```
>>--prependFullRow(--row--)----------------------><
```

Adds an item to this list-view at the first position in the list using a *LvFullRow* object.

Arguments:

The single argument is:

row [required]

A **`LvFullRow`** object that specifies the item to be added to the list-view.

Return value:

On success, returns the zero-based index of the added row. On error, -1 is returned.

Remarks:

When a full row object is used to add an item to the list view, all the subitems are also added. The *prependFullRow* method always inserts the new item as the first item in the list. Because of this, the item index in the item and in the subitems in the full row object are ignored. After the item is inserted, the item index is updated in the item and in the subitems is updated to the correct value of the newly added item.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example ...

```
lvItem = .LvItem~new(0, "Business manager", 6)
lvSub1 = .LvSubItem~new(0, 1, "Tom", 14)
lvSub2 = .LvSubItem~new(0, 2, "Sawyer", 26)
lvSub3 = .LvSubItem~new(0, 3, "ts@google.com", 11)

lvFullRow = .LvFullRow~new(lvItem, lvSub1, lvSub2, lvSub3, .true)
list~prependFullRow(lvFullRow)
```

15.85. previous

```
>>-aListControl~previous(--item--)---------------><
```

The previous method retrieves the item that precedes, or is to the left of, *item*.

Arguments:

The only argument is:

item

The number of the item at which the search is to start.

Return value:

The number of the previous item, or -1 in all other cases.

15.86. previousSelected

```
>>-aListControl~previousSelected(--item--)-------><
```

The previousSelected method retrieves the selected item that precedes, or is to the left of, *item*.

Arguments:

The only argument is:

item

The number of the item at which the search is to start. The specified item itself is excluded from the search.

Return value:

The number of the selected item, or -1 in all other cases.

15.87. redrawItems

```
>>-aListControl~RedrawItems--+------------------------+------><
                             +-(--first--+---------+--)-+
                                         +-,--last-+
```

The redrawItems method forces a list-view control to redraw a range of items.

Arguments:

The arguments are:

first

The number of the first item to be redrawn. The default is 0.

last

The number of the last item to be redrawn. The default is 0.

Return value:

0

The specified range of items was redrawn.

1

For all other cases.

15.88. removeExtendedStyle

```
                                         +--------------------+
                                         V                    |
>>-aListControl~removeExtendedStyle(--"----+-BORDERSELECT-----+--"--)-------><
                                           +-CHECKBOXES-------+
                                           +-DOUBLEBUFFER-----+
                                           +-FLATSB-----------+
                                           +-FULLROWSELECT----+
                                           +-GRIDLINES--------+
                                           +-HEADERDRAGDROP---+
                                           +-INFOTIP----------+
                                           +-LABELTIP---------+
                                           +-MULTIWORKAREAS---+
                                           +-ONECLICKACTIVATE-+
                                           +-REGIONAL---------+
                                           +-SIMPLESELECT-----+
                                           +-SUBITEMIMAGES----+
                                           +-TRACKSELECT------+
                                           +-TWOCLICKACTIVATE-+
                                           +-UNDERLINECOLD----+
                                           +-UNDERLINEHOT-----+
```

The **`removeExtendedStyle`** method removes one or more extended *styles* of a List-View control.

Arguments:

The only argument is:

style

The extended style(s) to be removed. This is one or more of the styles listed in the syntax diagram, separated by blanks. For an explanation of the different styles, refer to the *addExtendedStyle* method.

Return value:

The return values are:

0

Success.

-1

An (internal) problem with the control's window handle.

-3

The style argument did not contain any of the extended style keywords.

Example:

The following example removes the check box style from the list view.

```
listView = self~newListView(IDC_LIST)
if listView == .nil then return

listView~removeExtendedStyle("CHECKBOXES")
```

15.89. removeItemData

```
>>--removeItemData(--index--)--------------------><
```

Removes and returns the item data associated with the specified list-view item.

Arguments:

The single argument is:

index [required]

The zero-based index of the item whose data value is being removed.

Return value:

Returns the item data value associated with the specified item, or **`.nil`** if there is no data associated with the item.

Remarks:

The list-view control allows the user (the Rexx programmer) to associate a data value with any, or all, of the list-view items. The data value can be any ooRexx object. The data value is set using the *setItemData* method. The data value can be retrieved without removing it from a list-view item through the *getItemData* method. Storing a user value for each item can be useful in any number of ways. One specific use is in the *sortItems* method.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example comes from a program where the user can add items to a list-view and rearrange the items in any she wants. The user can also reset the order of the items to its original order when the program started up. To do that, the original index of each item is stored in the data value for each item when it is added to the list-view

Since the list-view control associates the data value with the index of an item, when the items are reset to their original order, the data value is removed from the index it is currently at. When the items are added back to the list-view in their original order the data value is then associated with the new item index.

```
::method initDialog
  expose list

  list = self~newListView(IDC_LV)
  ...

  items = self~getItems
  index = 0

  do item over items
    list~addRow(index, , item~text, item~fName, item~lName)
    item~origIndex = index

    list~setItemData(index, item)
    index += 1
  end


::method onReset
  expose list

  countItems = list~items
  items = .array~new(countItems)

  do i = 1 to countItems
    data = list~removeItemData(i - 1)
    items[data~origIndex + 1] = data
  end

  list~deleteAll

  do i = 1 to countItems
    data = items[i]

    list~addRow(i - 1, , data~text, data~fName, data~lName)
    list~setItemData(i - 1, data)
  end

return 0
```

15.90. removeStyle

```
                                 +-------------------+
                                 V                   |
>>-aListControl~removeStyle(--"----+-ICON---------+-+--"--)----><
                                   +-SMALLICON-----+
                                   +-LIST----------+
                                   +-REPORT--------+
                                   +-ALIGNLEFT-----+
                                   +-ALIGNTOP------+
                                   +-AUTOARRANGE---+
                                   +-ASCENDING-----+
                                   +-DESCENDING----+
                                   +-EDIT----------+
                                   +-HSCROLL-------+
                                   +-VSCROLL-------+
                                   +-NOSCROLL------+
                                   +-NOHEADER------+
                                   +-NOSORTHEADER--+
                                   +-NOWRAP--------+
                                   +-SINGLESEL-----+
                                   +-SHOWSELALWAYS-+
                                   +-SHAREIMAGES---+
                                   +-GROUP---------+
                                   +-HIDDEN--------+
                                   +-NOTAB---------+
                                   +-NOBORDER------+
```

The removeStyle method removes one or more window styles of a list-view control.

Arguments:

The only argument is:

style

The window styles to be removed, which is one or more of the styles listed in the syntax diagram, separated by blanks. For an explanation of the different styles, refer to *createListView*.

Return value:

0 if this method fails.

15.91. replaceExtendedStyle

```
                                                     +--------------------+
                                                     V                    |
>>-aListControl~replaceExtendedStyle(-oldStyle--,--"---+-BORDERSELECT-----+--"-)-><
                                                       +-CHECKBOXES-------+
                                                       +-DOUBLEBUFFER-----+
                                                       +-FLATSB-----------+
                                                       +-FULLROWSELECT----+
                                                       +-GRIDLINES--------+
                                                       +-HEADERDRAGDROP---+
                                                       +-INFOTIP----------+
                                                       +-LABELTIP---------+
                                                       +-MULTIWORKAREAS---+
                                                       +-ONECLICKACTIVATE-+
                                                       +-REGIONAL---------+
                                                       +-SIMPLESELECT-----+
                                                       +-SUBITEMIMAGES----+
                                                       +-TRACKSELECT------+
                                                       +-TWOCLICKACTIVATE-+
                                                       +-UNDERLINECOLD----+
                                                       +-UNDERLINEHOT-----+
```

The **`replaceExtendedStyle`** method removes some of the List-View control's extended *styles* and adds new extended styles. This is the equivalent of first using the **`RemoveExtendeStyle`** method and then using the **`addExtendedStyle`** method.

Arguments:

The arguments are:

oldStyle

The extended style(s) to be removed. This is one or more of the styles listed in the syntax diagram, separated by blanks. For an explanation of the different styles, refer to the *addExtendedStyle* method.

newStyle

The extended style(s) to be added. Again, this is one or more of the styles listed in the syntax diagram, separated by blanks. See the *addExtendedStyle* method for the style descriptions.

Return value:

The return values are:

0

Success.

-1

An (internal) problem with the control's window handle.

-3

One, or both, of the style arguments had none of the extended style keywords.

Example:

The following example removes the cold and hot underlining and adds check boxes to the list view.

```
listView = self~newListView(IDC_LIST)
if listView == .nil then return

removeStyle = "UNDERLINECOLD UNDERLINEHOT"
addStyle = "CHECKBOXES"

listView~replaceExtendedStyle(removeStyle, addStyle)
```

15.92. replaceStyle

```
                                                +-------------------+
                                                V                   |
>>-aListControl~replaceStyle(--oldStyle--,--"----+-ICON----------+-+--"--)-><
                                                  +-SMALLICON-----+
                                                  +-LIST----------+
                                                  +-REPORT--------+
                                                  +-ALIGNLEFT-----+
                                                  +-ALIGNTOP------+
                                                  +-AUTOARRANGE---+
                                                  +-ASCENDING-----+
                                                  +-DESCENDING----+
                                                  +-EDIT----------+
                                                  +-HSCROLL-------+
                                                  +-VSCROLL-------+
                                                  +-NOSCROLL------+
                                                  +-NOHEADER------+
                                                  +-NOSORTHEADER--+
                                                  +-NOWRAP--------+
                                                  +-SINGLESEL-----+
                                                  +-SHOWSELALWAYS-+
                                                  +-SHAREIMAGES---+
                                                  +-GROUP---------+
                                                  +-HIDDEN--------+
                                                  +-NOTAB---------+
                                                  +-NOBORDER------+
```

The replaceStyle method removes a window style of a list-view control and sets new styles.

Arguments:

The arguments are:

oldStyle
: The window style to be removed.

newStyle
: The new window styles to be set, which is one or more of the styles listed in the syntax diagram, separated by blanks. For an explanation of the different styles, refer to *createListView*.

Return value:

0 if this method fails.

Example:

The following example comes from a program where the user can switch between the different view styles in a list-view. The list-view is assigned an image list for both the small and normal icons. Radio buttons for the different views are connected to methods. When the user clicks one of the radio buttons, the method for that radio button uses **`replaceStyle`**() to remove all of the other view styles and add the specified style.

```
::method initDialog
  expose list

  self~newRadioButton(IDC_RB_ICON)~check
  self~connectButtonEvent(IDC_RB_REPORT, "CLICKED", onReport)
  self~connectButtonEvent(IDC_RB_LIST, "CLICKED", onList)
  self~connectButtonEvent(IDC_RB_ICON, "CLICKED", onIcon)
  self~connectButtonEvent(IDC_RB_SMALL_ICON, "CLICKED", onSmallIcon)
  list  = self~newListView(IDC_LV_VIEWS)

  imageList = .ImageList~create(.Size~new(16), COLOR24, 7, 0)

  imageList~add(.Image~getImage("iconList_16.bmp"))
  list~setImageList(imageList, SMALL)

  imageList = .ImageList~create(.Size~new(32), flags, 7, 0)

  imageList~add(.Image~getImage("iconList_32.bmp"))
  list~setImageList(imageList, NORMAL)

  self~populateList(list)

::method onReport
  expose list
  list~replaceStyle("LIST ICON SMALLICON", "REPORT")

::method onList
  expose list
  list~replaceStyle("REPORT ICON SMALLICON", "LIST")

::method onIcon
  expose list
  list~replaceStyle("LIST REPORT SMALLICON", "ICON")

::method onSmallIcon
  expose list
  list~replaceStyle("LIST ICON REPORT", "SMALLICON")

::method populateList private
  use strict arg list

  list~InsertColumnEx(0, "Title", 150)
  ...
```

15.93. scroll

```
                         +-0-+     +-0-+
>>-aListControl~scroll(--+---+--,--+---+--)------><
                         +-x-+     +-y-+
```

The Scroll method scrolls the content of a list view control.

Arguments:

The arguments are:

x

An integer value specifying the amount of horizontal scrolling. If the control is in icon, small-icon, or report view, this value specifies the number of pixels to be scrolled. If it is in list-view, this value specifies the number of columns to be scrolled. The default value is 0.

y

An integer value specifying the amount of vertical scrolling. If the control is in icon, small-icon, or report view, this value specifies the number of pixels to be scrolled. If it is in list-view, this value specifies the number of lines to be scrolled. The default value is 0.

Return value:

0

Scrolling was successful.

1
: Scrolling failed.

15.94. select

```
>>-aListControl~select(--item--)-----------------><
```

The select method selects an item.

Arguments:

The only argument is:

item
: The number of the item.

Return value:

0
: The item was selected.

-1
: You did not specify *item*.

1
: For all other cases.

15.95. selected

```
>>-aListControl~Selected-------------------------><
```

The selected method returns the number of the selected item.

Return value:

The number of the item selected last, or -1 in all other cases.

15.96. selectedItems

```
>>-aListControl~SelectedItems--------------------><
```

The selectedItems method determines the number of selected items in a list-view control.

Return value:

The number of selected items.

15.97. setColumnOrder

```
>>--setColumnOrder(--order--)--------------------><
```

Changes the column order of a list-view control to some other order. Typically this method might be used along with the *getColumnOrder*() method to save and restore the user's column order. But, it can also be used to programmatically change the column order for any reason.

Arguments:

The single required argument is:

order

An array containing the index numbers of the columns in their new order. The number at the first position in the array is the column index of the column that should be the left-most. The number at the second position in the array, and so on.

The array has to contain the same number of items as the number of columns with no empty indexes.

If non-valid column indexes are intermingled with valid column indexes, then the columns are re-ordered, but with no easily predictable pattern. If all the number are invalid column indexes, then the order of the columns remains unchanged. This behavior is the behavior of the underlying control, ooDialog has no control over it.

Return value:

The possible return values are:

`.true`

The method succeeded.

`.false`

Some error occurred. This is unlikely.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

In this example an application has two modes. One mode emphasizes the occupation of the people in a database, the other mode emphasizes the people. The order of the columns is occupation, name, ... for the first mode and name, occupation, ... for the second mode. When the mode of the application changes, the order of the columns also changes.

```
  ...
  list = self~newListView(100)

  columnNames = .array~of("Occupation", "Name", "Age", "Sex")
  list~addExtendedStyle("FULLROWSELECT GRIDLINES CHECKBOXES")

  list~insertColumn(0, columnNames[1], 95)
  list~insertColumn(1, columnNames[2], 95)
  list~insertColumn(2, columnNames[3], 95)
  list~insertColumn(3, columnNames[4], 35)
  ...

::method toggleColumnOrder private
  expose list

  order = list~getColumnOrder
  tmp = order[1]
  order[1] = order[2]
  order[2] = tmp
  list~setColumnOrder(order)
```

15.98. setColumnWidth

```
>>-aListControl~setColumnWidth(--column--+----------+--)--------><
                                         +-,--width-+
```

The setColumnWidth method sets the width of a column in a report or list view.

Arguments:

The arguments are:

column

The number of the column of which the width is to be set. 0 is the first column.

width

The width of the column, in pixels. If you omit this argument, the column is sized automatically to the widest item in the list.

Return value:

0

The column width was set.

-1

You did not specify *column*.

1

For all other cases.

Example:

The following example enlarges the selected column by 10:

```
::method OnColumnClick
  use arg id, column
  curList = self~newListView(102)
  curList~setColumnWidth(column,curList~columnWidth(column)+10)
```

15.99. setColumnWidthPX

```
>>--setColumnWidthPX(--colIndex--+-----------+--)---------------><
                                 +--,-width--+
```

Sets the width, in pixels, of a column in a list-view control that has the report or list style.

Arguments:

The arguments are:

colIndex [required]

The zero-based index of the column whose width is to be set.

width [optional]

A positive whole number specifying the width in pixels for the column. If, the list-view is in report view, one of the following keywords are also valid:

AUTO:

The list-view will automatically size the column.

AUTOHEADER:

Automatically sizes the column to fit the column text. When this value for the last column, its width is set to fill the remaining width of the list-view control.

Return value:

Returns 0 on success and 1 on error.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

Provided that the list-view is in report view, this example has the list-view automatically size the first two columns, and then size the last column to take up all the rest of the width in the control.

```
list = self~newListView(IDC_LV_SALES)

list~setColumnWidthPX(0, "AUTO")
list~setColumnWidthPX(1, "AUTO")
list~setColumnWidthPX(2, "AUTOHEADER")
```

15.100. setFullRowText

```
>>--setFullRowText(--row--)----------------------><
```

Modifies all, or some, of the text for a list-view item and its subitems using a *LvFullRow* object.

Arguments:

The single argument is:

row [required]

The *row* argument must either be a **`LvFullRow`** object, or the zero-based index of the list-view item whose text is to be modified.

LvFullRow object:

When the *row* argument is a **`LvFullRow`** object, the text of the list-view item is modified to match the text in the full row object. For the item and each subitem, the text is only modified, if that item or subitem in the *row* full row object has its item *mask*, or subitem *mask* attribute set to contain the TEXT keyword. If the item's, or subitem's, mask does not contain the TEXT keyword, the text in the list-view item is left unchanged.

Item index:

When the *row* argument is the index of the item to modify, the item *data* of the list-view item *must* have been assigned a **`LvFullRow`** object. The list-view item's text is then modified to match the text in the assigned full row object.

Return value:

Returns true on success, false on error.

Remarks:

Using an item index for the *row* argument can be thought of as a *sync* operation. It is assumed that the programmer has changed the value of the text attributes of the **`LvFullRow`** object that is assigned as the item data of the list-view item. Then, invoking the *modifyFullRow* method has the effect of syncing up the underlying list-view item's text with the full row object's text attributes. In this case, the text of the item and the text of every subitem is set to the corresponding text value the item data's full row object.

In contrast, when a **`LvFullRow`** object is used as the *row* argument, it is possible to only update the text for the item or subitems that need to be updated. The example below demonstrates this technique.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example constructs a full row object for a list-view that has 6 subitems. *currentItem* is the index of one of the list-view items, the item whose text might be modified. Note that the text and mask attributes of the item and subitems of the full row are not set. The current text in all the edit controls is collected and compared to the old values stored in the rec array. If it does not match, the text attribute in the full row is set to the value from the edit control. In this way, when ! setFullRowText() is invoked, only the text of the item or subitems in the list-view that have actually changed are updated:

```
lNameText   = lNameEdit~getText~strip
fNameText   = fNameEdit~getText~strip
...
zipText     = zipEdit~getText~strip

lvFullRow = .LvFullRow~new(.LvItem~new(currentItem))
do i = 1 to 6
  lvFullRow~addSubitem(.LvSubItem~new(currentItem, i))
end

if lNameText \== rec[2] then do
  lvFullRow~item~text = lNameText
end
if fNameText \== rec[3] then do
  lvFullRow~subItem(1)~text = fNameText
end
...
if zipText \== rec[8] then do
  lvFullRow~subItem(6)~text = zipText
end

list~setFullRowText(lvFullRow)
```

15.101. setHoverTime

```
>>-aListControl~setHoverTime(--+------+--)----------------------><
                               +-time-+
```

This method is used to change the *hover* time. The hover time only affects list-view controls that have the track select, one click activate, or two click activate extended list-view styles. See the *addExtendedStyle* for a description of these styles.

Arguments:

The only optional argument is:

time

Specify the hover time in milliseconds. If this argument is omitted, the hover time is set to the system default. A negative number will also set the hover time back to the system default.

Return value:

The previous hover time. A value of -1 indicates the previous hover time was the system default.

Example:

The following example adds the one click activate, the track select, and the full row select extended list-view styles to a **`ListView`**. The hover time is set to .4 seconds. If the user pauses his mouse over a row in the list-view for .4 of a second the row is automatically selected.

```
::method initDialog

  ...
  list = self~newListView(IDC_LIST)
  list~addExtendedStyle("ONECLICKACTIVATE FULLROWSELECT TRACKSELECT")

  oldTime = list~setHoverTime(400)
  ...
```

15.102. setImageList

```
>>--setImageList(--src-+---------------+-+----------+-+----------+-)-----------><
                       +-,-typeOrWidth-+ +-,-height-+ +-,-ilType-+
```

Assigns, or removes, the specified image list for the list-view control. Using **`.nil`** for the first argument removes the current image list if any. The list-view control uses image lists for a number of things, the large icons, the small icons, the state images, and the group header images. (ooDialog does not yet support list-view groups.)

Arguments:

The arguments are:

src [required]

The source for the image list. This can be an *ImageList* object to assign to the list-view, a bitmap *Image* object, or the file name of the bitmap. When *src* is a bitmap, the ooDialog framework internally creates an image list from the bitmap as explained in the Remarks section.

If this argument is the **`.nil`** object, the existing image list, if any, is removed.

typeOrWidth [optional]

type: When *src* is an **`ImageList`** object, this argument specifies which image list to assign, (or remove.) The following keywords are used to specify the type, case is not significant:

NORMAL	SMALL
STATE	

The default is NORMAL.

width: When *src* is a single bitmap, this argument can be used to specify the width of a single image in the bitmap. The default is the actual height of the bitmap.

height [optional]

This argument is only used when *src* is a single bitmap. In that case *height* specifies the height of an image in the bitmap. The default is to use the actual height of the bitmap.

ilType [optional]

This argument is only used when *src* is a single bitmap. It is used to specify which image list is being assigned. Its usage is exactly the same as is documented for the *type* role of the *typeOrWidth* argument above. If omitted, the default is NORMAL.

Return value:

The existing image list is returned, if there is one. Otherwise, the `.nil` object is returned.

Remarks

Note that the MSDN documentation states that the appropriate dimensions for the normal and small icons should be used. It has been observed that if the size of the small icons is not appropriate, there can be drawing problems with the icons in report view. Especially if check boxes are enabled. The *cxSmIcon* and *cySmIcon* attributes of the *SM* class reflect the dimensions that should be used for the small icons. The *cxIcon* and *cyIcon* attributes of the same class reflect the appropriate dimensions for the normal icons.

As noted above the *src* argument can be either an *ImageList* object, or a single bitmap. Using an image list object is the most flexible option. The programmer creates the image list in the manner most suitable to the program. When the program is enhanced or changed it is easy to change the image list.

When *src* is a single bitmap it is used in this way: A single bitmap is created that consists of all the images for the image list. All the images must be the same width and height. The images are arranged in the bitmap in a row, side-by-side, in the order they should appear in the image list. ooDialog uses this bitmap to create an *.ImageList* object and assigns it to the list-view. The *src* argument can be either the file name of the bitmap or a bitmap *Image* object.

Although using a single bitmap will allow the programmer to save a couple of lines of code, it has a number of disadvantages. The image list is created with the exact number of images in the bitmap and no extra room to add images. The image list is created as an unmasked image list, with a color of 8 bit per pixel. If the program is enhanced or changed in a way that requires even a slight change to the image list, the single bitmap would need to be re-created.

When using a single bitmap, if the system fails to create the image list from the details supplied by ooDialog, the method of informing the programmer of the error is not as robust as it is when the programmer creates the image list herself. However, ooDialog will set the *.systemErrorCode* for any cases where the operating system reports an error. Also note that when using a single bitmap, ooDialog will instantiate a *.ImageList* object. The programmer can retrieve this object using the *getImageList*() method and then manipulate it using the methods of the *ImageList* class.

Unless a list-view has the LVS_SHAREIMAGELISTS style, (corresponds to the SHAREIMAGES style in the *createListView*() method,) the list-view control will handle releasing the image list(s). When the share image lists style is used, ownership of the image list remains with the programmer. The *ImageList* and *Image* classes are used to manage image lists and images in ooDialog. The documentation on both classes discusses when and why the programmer may want to release image lists. The Image class documentation has the most detail on this subject.

Details:

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect arguments are detected.

Example:

This is a complete working example. It creates an image list using the application icons that ooDialog makes available for general use. The image list is assigned as the normal icon image list.

```
  dlg = .ImageDisplay~new
  if dlg~initCode == 0 then do
    dlg~constDir[IDC_LV_IMAGES] = 100
    dlg~create(30, 30, 320, 155, "ooDialog Icon Resources", , , "Tahoma")
    dlg~execute("SHOWTOP", IDI_DLG_OOREXX)
  end

::requires "ooDialog.cls"

::class 'ImageDisplay' subclass UserDialog

::method defineDialog

  self~createListView(IDC_LV_IMAGES, 10, 10, 300, 135, "ICON SINGLESEL");

::method initDialog

  names = .array~new()
  names[1] = "IDI_DLG_OODIALOG"
  names[2] = "IDI_DLG_APPICON"
  names[3] = "IDI_DLG_APPICON2"
  names[4] = "IDI_DLG_OOREXX"

  ids = .array~new()
  ids[1] = self~constDir[IDI_DLG_OODIALOG]
  ids[2] = self~constDir[IDI_DLG_APPICON]
  ids[3] = self~constDir[IDI_DLG_APPICON2]
  ids[4] = self~constDir[IDI_DLG_OOREXX]

  size = .Size~new(.SM~cxIcon, .SM~cyIcon)

  oodModule = .ResourceImage~new(self)
  icons = oodModule~getImages(ids, 'ICON', size)

  imageList = .ImageList~create(size, 'COLOR24 MASK', 4, 0)
  imageList~addImages(icons)

  list = self~newListView(IDC_LV_IMAGES)
  list~setImageList(imageList, 'NORMAL')

  do i = 1 to ids~items
    list~add(names[i] '('ids[i]')', i - 1)
  end
```

15.103. setItemData

```
>>--setItemData(--index--,--data--)--------------><
```

Sets the item data associated with the specified list-view item.

Arguments:

The arguments are:

index [required]

The zero-based index of the item the data value is associated with.

data [required]

The data value to store for the list-view item. This can be any Rexx object. (Recall that in ooRexx numbers and strings are also objects.)

Return value:

Returns true on success, false on error. An error is unlikely.

Remarks:

The list-view control allows the user (the Rexx programmer) to associate a data value with any, or all, of the list-view items. The data value can be any ooRexx object. The data value can be retrieved without removing it from a list-view item through the *getItemData* method, or it can be retrieved and removed through the *removeItemData*. Storing a user value for each item can be useful in any number of ways. One specific use is in the *sortItems* method.

For the *sortItems* method to work, every list-view item must have the item data set. When *sortItems* is invoked to have the ooDialog framework do the sorting internally, the item data object must be a *LvFullRow* object.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example stores a **`.Directory`** object with each list-view item when the item is added to the list-view. The **`.Directory`** object contains the text for the item and its subitems and additional information not shown in the code snippet:

```
::method initDialog
  expose list

  list = self~newListView(IDC_LV)
  ...

  items = self~getItems

  do item over items
    index = list~addRow(, , item~text, item~fName, item~lName)

    if index == -1 then do
      -- do some error routine
      return 0
    end

    list~setItemData(index, item)
  end
```

15.104. setItemPos

```
                                      +-0-+     +-0-+
>>-aListControl~setItemPos(--item--,--+---+--,--+---+--)--------><
                                      +-x-+     +-y-+
```

The setItemPos method moves an item to a specified position in a list-view control, which must be in icon or small-icon view.

Arguments:

The arguments are:

item

The number of the item.

x

: The x-coordinate of the new position of the upper left corner of the item, in view coordinates. The default is 0.

y

: The y-coordinate of the new position of the upper left corner of the item, in view coordinates. The default is 0.

Return value:

0

: The item was moved.

-1

: You did not specify *item*.

1

: For all other cases.

Note

Use *defListDragHandler* to support default dragging:

```
self~connectListViewEvent(104,"BEGINDRAG","mthDefListDragHandler")
```

15.105. setItemState

```
>>-aListControl~setItemState(--item--+------------------------------+--)-><
                                     |       +-----------------+    |
                                     |       V                 |    |
                                     +-,--"----+-CUT---------+-+--"-+
                                               +-NOTCUT------+
                                               +-DROP--------+
                                               +-NOTDROP-----+
                                               +-FOCUSED-----+
                                               +-NOTFOCUSED--+
                                               +-SELECTED----+
                                               +-NOTSELECTED-+
```

The setItemState method sets the state of a list view item.

Arguments:

The arguments are:

item

: The number of the item.

state

: The state of the item, which can be one or more of the following values, separated by blanks:

  CUT

  : The item is marked for a cut-and-paste operation.

NOTCUT
: The item cannot be used for a cut-and-paste operation.

DROP
: The item is highlighted as a drag-and-drop target.

NOTDROP
: The item is not highlighted as a drag-and-drop target.

FOCUSED
: The item has the focus and is therefore surrounded by the standard focus rectangle. Only one item can have the focus.

NOTFOCUSED
: The item does not have the focus.

SELECTED
: The item is selected. Its appearance depends on whether it has the focus and on the system colors used for a selection.

NOTSELECTED
: The item is not selected.

Return value:

0
: The state was set successfully.

-1
: You did not specify *item*.

1
: For all other cases.

15.106. setItemText

```
>>--setItemText(--item--,--+-----------+--,--text--)------------><
                           +--subitem--+
```

The setItemText method changes the text of a label, or the text of a subitem, of a list-view item.

Remember that in a list-view, items and columns use a 0-based index. Subitems use a 1-based index.

Arguments:

The arguments are:

item
: The index of the item.

subitem
: This method works in a similar manner as the *insert*() method. The use of the *subitem* argument controls whether the *text* argument applies to the label of the item or the text of a subitem. When *subitem* is omitted or 0, then the label of the item is set to *text*. When *subitem* is greater than 0, then the text of that subitem is set to *text*.

text
: The text for the item or a subitem, depending on the value of *subitem*.

Return value:

0
: The change was successful.

-1
: You did not specify *item*.

1
: For all other cases.

15.107. setToolTips

```
>>--setToolTips(--toolTip--)---------------------><
```

Sets the child *ToolTip* control used by this list-view.

Arguments:

The arguments are:

toolTip [required]
: The Rexx **`ToolTip`** object that represents the tool tip control the list-view should use.

Return value:

Returns the previous tool tip, as a Rexx **`ToolTip`** object, or the **`.nil`** object if there is no previous tool tip.

Details

Raises syntax errors when incorrect usage is detected.

15.108. setView

```
>>--setView(--view--)----------------------------><
```

Sets the view of this list-view.

Arguments:

The single argument is:

view [required]
: Exactly one of the following keywords that specifies which view to set. Case in not significant:

ICON	LIST
SMALLICON	REPORT

ICON
: Set the list-view to icon view. Each item appears as a full-sized icon with a label below it.

SMALLICON
: Set the list-view to small icon view. Each item appears as a small icon with the label to the right of it.

LIST
: Set the list-view to list view. Each item appears as a small icon with a label to the right of it. Items are arranged in columns.

REPORT
: Set the list-view to report view. Each item appears on its own line, with information arranged in columns.

Return value:

Returns the view of the list-view at the time this method was invoked. The view is returned as one of the keywords listed for the *view* argument.

Details

Requires Common Control *Library* version 6.0 or later.

Raises syntax errors when incorrect usage is detected.

Example:

This example shows a method that switches the view of the list-view to that specified by the index:

```
::method refreshView unguarded
  expose lv
  use strict arg index

  select
    when index == 1 then lv~setView("LIST")
    when index == 2 then lv~setView("REPORT")
    when index == 3 then lv~setView("ICON")
    when index == 4 then lv~setView("SMALLICON")
    otherwise return .false
  end
  -- End select

  return .true
```

15.109. smallSpacing

```
>>-aListControl~SmallSpacing---------------------><
```

The smallSpacing method determines the spacing between items in a small-icon list-view control.

Return value:

The amount of spacing between the items.

15.110. snapToGrid

```
>>-aListControl~SnapToGrid-----------------------><
```

The snapToGrid method snaps all icons to the nearest grid position.

Return value:

0
: The items were snapped.

1
: For all other cases.

15.111. sortItems

```
>>--sortItems(--methodName--+----------+--)------><
                            +-,-param--+
```

The *sortItems* method causes the list-view control to sort its items using the specified comparison method. The comparison method can be a method in the Rexx dialog, or a method name that signals the ooDialog framework to use an internal comparison method written in the native API.

Arguments:

The arguments are:

methodName [required]
: *methodName* is either the name of a method in the Rexx dialog, (referred to as a Rexx sort,) or the special method name of: *InternalListViewSort*, (referred to as an internal sort.)

  When *methodName* is *InternalListViewSort* then the ooDialog framework performs the sorting work internally. This can be significantly faster, for very large lists, but there are a number of restrictions to this sort. See the remarks section for the details.

  When *methodName* is not *InternalListViewSort*, then it names a comparison method in the Rexx dialog that is used by the list-view to sort its items. The details of how the Rexx method is implemented are in the remarks section.

param [optional]
: For the **Rexx sort**, the *param* argument can be any object the programmer wants to use. If the argument exists, it is passed as the third argument to the *methodName* comparison method. If it is not specified, **`.nil`** is passed as the third argument.

  For the **internal sort**, the *param* argument can be a **`Directory`** object that is used to change the internal sort parameters. If *param* is omitted in this case, the sort is performed on the list-view item text, as a case-sensitive ascending sort. If not omitted, the following indexes in the **`Directory`** object are checked and alter the sort as described:

  ASCENDING:
  : If present, the index must be true or false. When true, the sort is an ascending sort, if false the sort is a descending sort. The default if the index is missing, is true.

  CASELESS:
  : If present, the index must be true or false. When true, the sort is case-insensitive, if false the sort is case-sensitive. The default if the index is missing, is false.

  COLUMN:
  : If present, the index must be a non-negative number, within the range of 0 to the number of subitem columns in the list-view. Recall that the first subitem is in column 1, the second subitem is in column 2, and so on. The value of the COLUMN index specifies which text the sort is performed on. A value of 0 specifies that the sort is performed on the item text. Non-zero values specify that the sort is performed on the text of the subitem specified.

I.e., a value of 2 specifies the sort is performed on the text of subitem 2. If omitted, the default value is 0.

Return value:

Returns true on success, false on error. An error is unlikely.

Remarks:

The list-view control allows the user (the Rexx programmer) to associate a user *data* value with any, or all, of the list-view items. The data value can be any ooRexx object. For the *sortItems* method to work correctly, the programmer must associate a data value with each list-view item.

The specifics on the item data value and what the programmer needs to do are dependent on the type of sort specified. See the relevant section below for a Rexx sort or an internal sort.

Rexx Sort

When the Rexx sort is performed, as the list-view control needs to compare two items it will invoke, through the ooDialog framework, the comparison method named by the *methodName* argument in the Rexx dialog. The return from the Rexx method determines how the items are sorted.

The comparison method must return a whole number and this value is passed on to the list-view control. While the list-view is sorting, the list is not stable and the comparison method must not invoke any **`ListView`** methods. If methods of the **`ListView`** object are invoked, the results of the sort are unpredictable. The comparison method should be coded as follows:

```
::method compare
  use arg data1, data2, param
```

Arguments:

The comparison method receives 3 arguments:

data1

The *data value* assigned to the first list-view item to be compared. This data value is the value the programmer assigned to the item using the *setItemData* method.

data2

The *data value* assigned to the second list-view item to be compared.

param

The *param* argument passed to the *sortItems* method, if it exists. If it was omitted, the param argument will be the **`.nil`** object.

Return:

The return must be a whole number. Less than 0 places the first list item before the second item, greater than 0 places the first item after the second. Return 0 if the two items are equivalent. The programmer determines what number to return based on the 2 item data values passed to the method.

Internal Sort

When an internal sort is used, the work of the sort is done in the native API implementation code. In essence, within the ooDialog framework. To use this sort, the item data assigned to each list-view item must be a *LvFullRow* object. The item data can be assigned to each item either through the *setItemData* method, or during the *instantiation* of the **`LvFullRow`** object. The sort will not work correctly if any item in the list-view does not have a **`LvFullRow`** object assigned as its item data value.

The sort is performed on the text of the item, or the text of the subitems of the item. By default, the sort is an ascending, case-sensitive, sort of the text of the item. The *param* argument can be used to change this to a sort on the text of any of the subitems of the item, as explained above. Ascending or descending sorts, as well as case-sensitive or case-insensitive sorts can also be specified using the *param* argument.

Details

Raises syntax errors when incorrect usage is detected.

Rexx Sort Example:

This example shows one approach to sorting the list items on either a first name or last name, and either ascending or descending using the Rexx sort:

```
::method initDialog
  ...
  -- Get the list-view object
  list = self~newListView(IDC_LV)

  -- Get the data value for each list item.
  -- names is an array of .directory objects.
  -- Each .directory object has a fName and a
  -- lName index
  names = self~getFirstLastNames

  do i = 1 to names~items
    list~setItemData(i - 1, names[i])
  end
  ...

::method sortList unguarded
  use strict arg ascending, column, list

  -- column is the column to sort on. Column 1 is
  -- the first name, column 2 is the last name.

  if ascending then list~sortItems(sortUp, column)
  else list~sortItems(sortDown, column)

::method sortUp unguarded
  use arg data1, data2, param

  if param == 1 then return data1~fName~compareTo(data2~fName)
  else return data1~lName~compareTo(data2~lName)

::method sortDown unguarded
  expose field
  use arg data1, data2, param

  if param == 1 then return data2~fName~compareTo(data1~fName)
  else return data2~lName~compareTo(data1~lName)
```

Internal Sort Example:

This example shows the basic steps needed to set up an internal sort. In the example, a descending, case-insensitive sort is done on the text of subitem 3:

```
::method initDialog
    expose list
    ...

    list = self~newListView(IDC_LV_CONTACTS)
    items = self~createRows
    do r over items
        list~addFullRow(r)
    end

::method createRows private

    ...
        -- As each full row object is instantiated, the last argument
        -- is set to .true to indicate the full row object should be
        -- set as the item data of the list-view item:

        rows[i] = .LvFullRow~new(lvi, lvsi1, lvsi2, lvsi3, .true)

    return rows

-- The onSort() method is connected to a 'Sort' push button
::method onSort unguarded
    expose list

    d = .directory~new
    d~column    = 3
    d~ascending = .false
    d~caseless  = .true

    list~sortItems('InternalListViewSort', d)
```

15.112. spacing

```
>>-aListControl~Spacing--------------------------><
```

The spacing method determines the spacing between items in an icon list-view control.

Return value:

The amount of spacing between the items.

15.113. stringWidth

```
>>-aListControl~stringWidth(--text--)------------><
```

The stringWidth method determines the width of a specified string using the current font of the list-view control.

Arguments:

The only argument is:

text

The text string of which the width is to be determined.

Return value:

The string width, or -1 if you did not specify a *text*, or 0 in all other cases.

15.114. stringWidthPX

```
>>--stringWidthPX(--text--)----------------------><
```

Determines the width, in pixels, of the specified string using the list-view's current font.

Arguments:

The single arguments is:

text

The text whose width is desired.

Return value:

Returns the exact width of the text in pixels, or 0 on error.

Remarks:

The *stringWidthPx* method returns the exact width, in pixels, of the specified string. If the returned width is used as the column width in the *setColumnWidthPX* method, the string will be truncated. To get the column width that can contain the string without truncating it, the programmer must add padding to the returned width.

15.115. textBkColor

```
>>-aListControl~TextBkColor----------------------><
```

The textBkColor method retrieves the background color of the text in a list-view control.

Return value:

The *color palette* index specifier (0 to 18).

15.116. textBkColor=

```
>>-aListControl~textBkColor=(--color--)----------><
```

The textBkColor= method sets the background color for the text in a list-view control.

Arguments:

The only argument is:

color

The new background color for text. Specify the *color palette* index specifier (0 to 18).

15.117. textColor

```
>>-aListControl~TextColor------------------------><
```

The textColor method retrieves the text color of a list-view control.

Return value:

The *color palette* index specifier (0 to 18).

15.118. textColor=

```
>>-aListControl~textColor=(--color--)------------><
```

The textColor= method sets the text color of a list-view control.

Arguments:

The only argument is:

color

The new text color. Specify the *color palette* index specifier (0 to 18).

Example:

The following example sets the text color of a list control to light blue:

```
::method LightBlue
  curList = self~newListView(104)
  curList~bkColor = 9
  curList~update
```

15.119. uncheck

```
>>-aListControl~uncheck(--index--)---------------><
```

The **uncheck** method removes the check mark for a list-view item. This method is only valid for list-view controls that have the check boxes extended style. (See the *addExtendedStyle* method for a description of the check boxes extended style.)

Arguments:

The only argument is:

index

The index of the item where the check box check is to be removed.

Return value:

The return values are:

0

Success.

-1

An (internal) problem with the control's window handle.

-2

The list-view control does not have the check boxes extended style.

-3

The index is invalid.

Example:

The following example is from a doctor's office custom accounting application. A list box is filled with information for every patient. One button in the application checks every patient with an outstanding balance, they will be mailed a statement for that balance. The doctor does not believe in sending billing statements to patients that have a very low balance. However what a "low" balance is depends on how he is feeling at statement time. Therefore the application allows the office manager to dynamically remove some patients from the statement list.

```
::method onDeferBilling
  expose billingList

  text = self~newEdit(IDC_EDIT_MINAMOUNT)~getText
  minimum = self~decimalNumber(text)

  if minimum > 0 then do i = 0 to billingList~items
    if billingList~isChecked(i) then do

      -- Column 6 in the list is the patient's current balance.
      owes = billingList~itemText(i, 6)
      if self~decimalNumber(owes) <= minimum then billingList~uncheck(i)

    end
  end
```

15.120. uncheckAll

```
>>-aListControl~UncheckAll-----------------------><
```

The **uncheckAll** method clears the check mark for every list-view item. This method is only valid for list-view controls that have the check boxes extended style. (See the *addExtendedStyle* method for a description of the check boxes extended style.)

Arguments:

This method takes no arguments.

Return value:

The return values are:

0
: Success.

-1
: An (internal) problem with the control's window handle.

-2
: The list-view control does not have the check boxes extended style.

Example:

The following example is from a program that has a push button that allows the user to remove the check mark from all the check boxes:

```
::method onUncheckAll

  list = self~newListView(IDC_LIST_REPUBLICANS)
  list~uncheckAll
```

15.121. updateItem

```
>>-aListControl~updateItem(--item--)-------------><
```

The updateItem method updates a list-view item.

Arguments:

The only argument is:

item
: The number of the item to be updated.

Return value:

0
: The item was updated.

-1
: You did not specify *item*.

1
: For all other cases.

15.122. LvCustomDrawSimple Class

A **`LvCustomDrawSimple`** object is used when a *ListView* control is registered for *simple custom* draw. A **`LvCustomDrawSimple`** object is passed to the CUSTOMDRAW event handler. It supplies both information to the event handler and returns information back to the Windows control. The attributes of the object convey the information both ways.

When the ooDialog framework receives information from the Windows custom draw control, it instantiates a **`LvCustomDrawSimple`**object, assigns values to the attributes of the object, and sends the object to the Rexx dialog's event handler. The programmer uses the assigned values to determine what action to take and then assigns values to the object's attributes to convey information back to the Windows control. The **`LvCustomDrawSimple`** object is specific to the **`ListView`** class. Other ooDialog controls that support custom draw have their own class that performs a similar function, but whose attributes are specific to that control. For instance, the *TreeView* class uses the *TvCustomDrawSimple* for its CUSTOMDRAW event handler.

15.122.1. CustomDraw Event Handler

When the list-view control is registered for *simple* custom draw, the event handler is invoked when the draw stage is item prepaint. Depending on the response from the event handler to the item prepaint notification, the event handler will also be invoked when the draw stage is subitem prepaint. For simple custom draw, the ooDialog framework handles the other draw stages internally. It makes the appropriate response to the dialog control so that the dialog control will send the item prepaint notifications. It is then up to the programmer to request that the control send the subitem prepaint notifications.

The single argument passed to the event handler is a **`LvCustomDrawSimple`** object. The event handler examines the values of the attributes of the object to determine the information sent by the underlying list-view control. The event handler then assigns values to the object that specify how the list-view control should draw the item, or subitem. Finally, the programmer returns true to have the updated information passed on to the list-view, or false to have the list-view draw the item as it normally would.

```
::method onCustomDraw unguarded
```

```
  use arg lvcdSimple

  return boolean
```

Arguments:

The event handling method receives 1 argument:

lvcdSimple

A **`LvCustomDrawSimple`** object whose attributes are used to convey information back and forth between the underlying list-view control and the event handler.

Return:

Return true to indicate that the attributes in the **`LvCustomDrawSimple`** object are to be used by the list-view when drawing the item or subitem. Return false to indicate the list-view should draw the item itself. Returning false is the equivalent of using the *CDRF_DODEFAULT* value for the response to the list-view.

Remarks:

The key to using custom draw is to examine the attributes of the **`LvCustomDrawSimple`** object sent as the argument to the event handler to determine which item is about to be drawn, and then set attributes in the object to customize the drawing of that item.

As an example, say the goal was to shade every other row in the list-view when in report mode. The event handler will be invoked before each item is to be painted. The programmer would examine the *item* attrbute to determine if it was an odd or even row. If the row was even, the color attributes, *clrTextBk* and *clrText*, would be set to the desired colors to shade the row. The *reply* attribute would be set to a response value that indicates to the list-view is to use the information in the object to draw the item, and the event handler would return true. If the row was odd, the event handler would simply return false to indicate that the list-view should draw the item itself. The following code snippet demonstrates this:

```
::method init
    expose rowYellow

    forward class (super) continue

    self~customDraw
    self~customDrawControl(IDC_LV_CUSTOM, 'ListView', onCustomDraw)

    rowYellow   = self~RGB(245, 245, 127)
    ...

::method onCustomDraw unguarded
    expose rowYellow
    use arg cdInfo

    item = cdInfo~item

    if (item // 2) == 0 then do
        cdInfo~clrText    = self~CLR_DEFAULT
        cdInfo~clrTextBk  = rowYellow
        cdInfo~reply      = self~CDRF_NOTIFYITEMDRAW
        return .true
    end
    else do
        return .false
    end
```

Example

The following example is an expansion of the example in the remarks section. It has a list-view that has check boxes. If an item is checked then a new font is set for that row, along with different colors. Othewise, alternating rows are shaded with a different color:

```
::method onCustomDraw unguarded
    expose list textRed textBlack textBlue rowGreen rowLiteBlue rowYellow checkedFont
    use arg cdInfo

    item = cdInfo~item

    if list~isChecked(item) then do
        cdInfo~clrText   = textBlue
        cdInfo~clrTextBk = rowLiteBlue
        cdInfo~font      = checkedFont
    end
    else if (item // 2) == 0 then do
        cdInfo~clrText   = textRed
        cdInfo~clrTextBk = rowYellow
    end
    else do
        cdInfo~clrText   = textBlack
        cdInfo~clrTextBk = rowGreen
    end

    cdInfo~reply = self~CDRF_NEWFONT
    return .true
```

15.122.2. Method Table

The following table lists the class and instance methods of the **`LvCustomDrawSimple`** class:

Table 15.3. LvCustomDrawSimple Class Method Reference

Method	Description
Class Methods	**Class Methods**
new	The Rexx programmer can not instantiate a **`LvCustomDrawSimple`**object, the ooDialog framework instantiates these objects
Attribute Methods	**Attribute Methods**
clrText	Reflects a custom color that the text of the list-view item should be drawn with.
clrTextBk	Reflects a custom color that the text background of the list-view itme should be drawn with.
drawStage	Reflects the draw *stage* of the current paint cycle.
font	Reflects a custom font that the list-view should use for the text.
id	Reflects the resource ID of the list-view control sending the custom draw event notification message.
item	Reflects the 0-based index of the list-view item that needs to be drawn.
itemData	Reflects the *item data* for the list-view item that needs to be drawn.
reply	Reflects the *CDRF_** response value that is used to reply to the event notification.
subItem	Reflects the index of the subitem that needs to be drawn.

15.122.3. new (Class Method)

```
>>--new-----------------------------------------><
```

The **`LvCustomDrawSimple`** class can not be instantiated by the Rexx programmer. When needed, the ooDialog framework instantiates a **`LvCustomDrawSimple`** object and passes it to the dialog's custom draw *event* handler.

15.122.4. clrText (Attribute)

```
>>--clrText-------------------------------------><

>>--clrText-=-varName---------------------------><
```

Reflects a custom color that the text of the list-view item should be drawn with. This attribute is information sent back to the list-view control.

clrText get:
This is a set-only attribute, it is information to be sent back to the list-view control.

clrText set:
Set this attribute to the color value for the text of the item.

Remarks:
To construct the proper color value use the *rgb* method of the **`CustomDraw`** class. The *CLR_DEFAULT* value can also be used to have the item text be drawn in the default list-view color.

Details
Raises syntax errors when incorrect usage is detected.

Example:
In this example snippet of code the *clrText* attribute is set to a reddish color and the background color is set to a yellowish color:

```
  cdInfo~clrText   = self~RGB(247,   7,  59)
  cdInfo~clrTextBk = self~RGB(245, 245, 127)
```

15.122.5. clrTextBk (Attribute)

```
>>--clrTextBk-----------------------------------><

>>--clrTextBk = varName-------------------------><
```

Reflects a custom color that the text background of the list-view itme should be drawn with. This attribute is information sent back to the list-view control.

clrTextBk get:
This is a set-only attribute, it is information to be sent back to the list-view control.

clrTextBk set:

Set this attribute to a custom color that should be used for the text background when the item is drawn.

Remarks:

To construct the proper color value use the *rgb* method of the **`CustomDraw`** class. The *CLR_DEFAULT* value can also be used to have the item text background drawn in the default list-view color.

Details

Raises syntax errors when incorrect usage is detected.

Example:

In this example snippet of code the *clrText* attribute is set to a reddish color and the background color is set to a yellowish color:

```
cdInfo~clrText   = self~RGB(247,   7,  59)
cdInfo~clrTextBk = self~RGB(245, 245, 127)
```

15.122.6. drawStage (Attribute)

```
>>--drawStage------------------------------------><

>>--drawStage = varName--------------------------><
```

Reflects the draw *stage* of the current paint cycle. This attribute is information sent from the list-view control.

drawStage get:

Returns the draw stage constant value as sent by the list-view control.

drawStage set:

This is a get-only attribute. It is information sent from the list-view control and can not be changed.

Remarks:

In *simple* custom draw, the ooDialog framework handles many of draw stage notifications internally and the event handler only gets invoked for 2 of the draw stages. Because of this, the draw stage value will always be either *CDDS_ITEMPREPAINT* or *CDDS_SUBITEMPREPAINT*. In addition, the value will only be CDDS_SUBITEMPREPAINT if the programmer has requested subitem notifications by setting the *reply* attribute to *CDRF_NOTIFYSUBITEMDRAW* in response to an item prepaint notification.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example is from a program that uses a custom font and color for the entire row, if the item is checked. If not checked then it uses custom draw for each subitem. The draw stage will either be CDDS_ITEMPREPAINT or CDDS_SUBITEMPREPAINT. If it is CDDS_ITEMPREPAINT, then the event hander determines if the item is checked.

When the item is checked, the event handler sets the custom color and font. Then, because it does not return CDRF_NOTIFYSUBITEMDRAW in the response, the list-view draws the entire row in the custom color and font, and does *not* send any subitem notifications. When the item is not checked, CDRF_NOTIFYSUBITEM is set in the response and this causes the list-view to send event notifications for each subitem:

```
...

if cdInfo~drawStage == self~CDDS_ITEMPREPAINT then do
    if list~isChecked(item) then do
        cdInfo~clrText   = textBlue
        cdInfo~clrTextBk = rowLiteBlue
        cdInfo~font      = checkedFont
        cdInfo~reply     = self~CDRF_NEWFONT
    end
    else do
        cdInfo~reply = self~CDRF_NOTIFYSUBITEMDRAW
    end
    return .true
end

...
```

15.122.7. font (Attribute)

```
>>--font-----------------------------------------><

>>--font = varName-------------------------------><
```

Reflects a custom font that the list-view should use for the text. This attribute is information sent back to the list-view control.

font get:

This is a set-only attribute, it is information to be sent back to the list-view control.

font set:

Set this attribute to the *handle* of a font to used for the text of the item.

Remarks:

To have the font drawn in the default font of the list-view simply do not set the *font* attribute to anything. For example, if the color for the font should be changed, but not the font itself, then the programmer should not set this attribute.

Use the *createFontEx* method to get the handle of a font. Microsoft recommends that the response code when a new font is to be used include *CDRF_NEWFONT* value. If the programmer also needs subitem notifications and is changing the font, then the CDRF_NEWFONT code should be combined with the *CDRF_NOTIFYSUBITEMDRAW* value. The *or* method of the **DlgUtil** class can be used to combine the values.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows the *font* attribute being set and the response codes for new font and subitem notify being combined:

```
checkedFont = self~createFontEx("Courier New", 10)
```

```
  ...

  item = cdInfo~item

  if list~isChecked(item) then do
      cdInfo~clrText   = self~RGB( 17,   5, 250)
      cdInfo~clrTextBk = self~RGB(115, 245, 186)
      cdInfo~font      = checkedFont
      cdInfo~reply     = .DlgUtil~or(self~CDRF_NEWFONT, self~CDRF_NOTIFYSUBITEMDRAW)
  end

  ...
```

15.122.8. id (Attribute)

```
>>--id-------------------------------------------><

>>--id = varName---------------------------------><
```

Reflects the resource ID of the list-view control sending the custom draw event notification message. This attribute is information sent from the list-view control.

id get:

Returns the numeric *resource ID* of the list-view control that sent the custom draw event notification.

id set:

This is a get-only attribute. It is information sent from the list-view control and can not be changed.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example uses custom draw with a list-view and a tree-view. It uses the same event handler, *onCustomDraw* for both controls and then use the *id* attribute to determine which control has sent the event notification:

```
  self~customDraw
  self~customDrawControl(IDC_LV_STATS, 'ListView', onCustomDraw)
  self~customDrawControl(IDC_TV_PLAYERS, 'TreeView', onCustomDraw)

  ...

::method onCustomDraw unguarded
    use arg customDrawInfo

    if customDrawInfo~ID == .constDir[IDC_LV_STATS] then
        return self~customDrawStats(customDrawInfo)
    else
        return self~customDrawPlayers(customDrawInfo)
```

15.122.9. item (Attribute)

```
>>--item-----------------------------------------><
```

```
>>--item = varName-------------------------------><
```

Reflects the 0-based index of the list-view item that needs to be drawn. This attribute is information sent from the list-view control.

item get:

Returns the item index that the event notifications is for.

item set:

This is a get-only attribute. It is information sent from the list-view control and can not be changed.

Remarks:

Use the item index to determine exactly which item or subitem is going to be painted.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shades every other row in the list-view. It uses the *item* attribute to determine if the row is an even or odd row:

```
::method onCustomDraw unguarded
    expose rowYellow
    use arg cdInfo

    if (cdInfo~item // 2) == 0 then do
        cdInfo~clrText   = self~CLR_DEFAULT
        cdInfo~clrTextBk = rowYellow
        cdInfo~reply     = self~CDRF_NOTIFYITEMDRAW
        return .true
    end
    else do
        return .false
    end
```

15.122.10. itemData (Attribute)

```
>>--itemData-------------------------------------><

>>--itemData = varName---------------------------><
```

Reflects the item *data* for the list-view item that needs to be drawn. This attribute is information sent from the list-view control.

itemData get:

Returns the item data object for the item about to be drawn, or **`.nil`** if no item data was set for the item.

itemData set:

This is a get-only attribute. It is information sent from the list-view control and can not be changed.

Remarks:

The list-view allows the programmer to associate a value with any list-view item. This item data value is sent by the list-view in the event notification information. The programmer can set the item data for a list-view item in several ways. The *setItemData* of the *ListView* class, the *itemData* argument of the *new* method of the *LvItem* class, or the *itemData* attribute of the **`LvItem`** class are all used to set the item data value for an item.

The *itemData* attribute could be used to help decide how to custom draw the list-view item or subitem, see the example below.

Details

Raises syntax errors when incorrect usage is detected.

Example:

In this example the color to draw the text for each item is stored in the item data for the item. This makes for a very simple event handler, the text color is just set to whatever value the item data is:

```
::method onCustomDraw unguarded
    use arg lvcds

    lvcds~clrText   = lvcds~itemData
    lvcds~clrTextBk = self~CLR_DEFAULT
    lvcds~reply     = self~CDRF_NEWFONT

    return .true
```

15.122.11. reply (Attribute)

```
>>--reply----------------------------------------><

>>--reply = varName------------------------------><
```

Reflects the *CDRF_** response value that is used to reply to the event notification. This attribute is information sent back to the list-view control.

reply get:

This is a set-only attribute, it is information to be sent back to the list-view control.

reply set:

The *reply* attribute should be set to one or a combination, of the CDRF_* constant values.

Remarks:

The *reply* attribute can be set to any valid combination of CDRF_* values. However, in *simple* custom draw only a few values make any sense. These are CDRF_NOTIFYITEMDRAW, CDRF_NOTIFYSUBITEMDRAW, and CDRF_NEWFONT.

Rather than set *reply* to CDRF_DODEFAULT and returning true from the event handler, the programmer should just return false from the event handler. Using any of the other CDRF_* values, in simple custom draw, will have no effect in the Rexx program and will cause the list-view control to send unnecessary event notifications.

The only combination of CDRF_* values that make sense is combining CDRF_NEWFONT and CDRF_NOTIFYSUBITEMDRAW. And, this is really only needed if the font is being changed and subitem notifications are desired. If only the colors are changed when subitem notifications are desired, then it is sufficient to use CDRF_NOTIFYSUBITEMDRAW by itself. Use the *or* method to combine two values if needed.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows setting the *reply* attribute using both the new font and notify subitem draw values:

```
customDraw~reply = .DlgUtil~or(self~CDRF_NEWFONT, self~CDRF_NOTIFYSUBITEMDRAW)
```

15.122.12. subItem (Attribute)

```
>>--subItem-------------------------------------><

>>--subItem = varName---------------------------><
```

Reflects the index of the subitem that needs to be drawn. This attribute is information sent from the list-view control.

subItem get:

Returns the subitem index that needs to be drawn. The value only has meaning if the *drawStage* attribute is *CDDS_SUBITEMPREPAINT*

subItem set:

This is a get-only attribute. It is information sent from the list-view control and can not be changed.

Remarks:

Experimentation has shown that when subitem notifications have been requested, when the CDDS_ITEMPREPAINT draw stage notification is sent, the *subItem* value is not always 0. This would indicate that the list-view control only bothers to set the *subitem* value for the CDDS_SUBITEMPREPAINT notification.

When the draw stage is CDDS_SUBITEMPREPAINT, if the item itself is being drawn, then *subItem* will be 0. Otherwise, *subItem* will be 1 for the first subitem, 2 for the second subitem, and so on.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example is a snippet of code from a program where the second subitem in a list-view is a number. If the number is negative, it is drawn in red text and if it is not negative, is is drawn in black text:

```
if customDrawInfo~subItem == 2 then do
    number = list~itemText(customDrawInfo~item, 2)
    if number < 0 then
        customDrawInfo~clrText = textRed
    else
        customDrawInfo~clrText = textBlack
end

...
```

15.123. LvFullRow Class

A **`LvFullRow`** object is used in ooDialog to represent a list-view item and all subitems of that item. A **`LvFullRow`** object is composed of a *LvItem* object whose attributes reflect the list-view item and zero or more *LvSubItem* objects that reflect the subitems of the list-view item in the *underlying* list-view.

Normally, the subitems of a list-view item are only thought about when the list-view is in report view. However, once added to an item, the subitems exist whatever view the list-view is in. A full row object can be used to add or insert list-view items along with all the subitems of the item, or to query information about a list-view item and its subitems.

`LvFullRow` objects also provide the infrastructure that allows the ooDialog framework to *internally* sort list-view items.

15.123.1. Internal Sorting Considerations

The *sortItems* method is used to sort list-view items. One option of this method is to have the ooDialog framework implement the sort internally. The internal sort can be significantly faster, for large lists, than implementing the sort in Rexx. However, the internal sorting is dependent on the **`LvFullRow`** class. There are a few things the Rexx programmer needs to be aware of that effect the ability of the ooDialog framework to do an internal sort.

Any sorting of the list-view items, whether it is through a sort implemented in Rexx or the internal sort, is dependent on *setting* a value for the item data of every list-view item. For the internal sort to work, the item data value must be a **`LvFullRow`** object. There are several approaches that could be used to assign a full row object to the item data of every list-view item. One approach would be to iterate through all the list-view items, for each item, instantiate a full row object for the item, and use the *setItemData* method to assign the object to the item.

However, full row objects can also be used to add items to the list-view through one the *addFullRow*, *insertFullRow*, or *prependFullRow* methods. A second approach is to add all list-view items using full row objects and have the ooDialog framework set the full row object as the item data when the item is added to the list-view. To use this technique, the programmer should set the *sorting* argument to true when *instantiating* the full row object.

In a way, a full row object is a snapshot of the information the list-view control maintains about a specific item in the list. This information can change over time as the application is used. For instance, the text of an item could be changed. When a full row object is assigned as the item data of a list-view item, the ooDialog framework does its best to keep the full row object updated with any changes made to the information of the list-view item. But, there is one scenario where this is not always possible. The Rexx programmer needs to be aware of this and take the proper steps to keep the full row object in synch with the list-view item.

When a subitem column in the list-view is *inserted* or *deleted*, the list-view control adds or deletes the subitems of all items that corresponds to the column. Every item in a list-view always has the same number of subitems. If all items in the list-view have a **`LvFullRow`** object set as the item data, when a column is deleted, the full row object will have one too many subitems. The same if a column is added, the full row object will have no knowledge of the new subitem. **Note** that there is no problem the very first time a column is inserted in a list-view and this only applies to *subitem* columns. The information for a list-view item is not deleted by deleting its column, column 0. In addition, this discussion only concerns the case where **`LvFullRow`** objects have been assigned to the item data of each item to facilitate internal sorting.

Most ooDialog programs that use a list-view do not involve inserting or removing columns after the list-view items have been added. But, for programs that do *and* use **`LvFullRow`** objects to allow internal sorting, the programmer must take an additional step when a column is inserted or deleted.

This ensures the integrity of the full row objects. ooDialog provides 3 methods to ensure integrity when a column is inserted or deleted:

The fixFullRowColumns method:

ooDialog provides the *fixFullRowColumns* method of the **`ListView`** which can be used to bring all the full row objects back in sync. It must be invoked after each individual column insert and delete to work correctly. That is, the programmer can not delete 2 columns and then invoke the *fixFullRowColumns* twice. The method is invoked with the column index that was removed or added and a second argument that signals whether the specified column was inserted. The following code snippet gives an example:

```
list~insertColumnPX(2, "Phone Numbers", 155, "RIGHT")
list~fixFullRowColumns(2, .true)
```

The adjustFullRows argument:

The methods that insert or delete columns in the list-view have an optional second argument, *adjustFullRows*. If this argument is true, then the ooDialog framework is signaled to fix up the full row objects internally when the column is added or deleted. The default for the argument is false. This is perhaps the easist process for the programmer to use. The above example would be altered like this:

```
list~insertColumnPX(2, "Phone Numbers", 155, "RIGHT", .true)
```

The reason that the default here is false is that with very large lists, the time to fix up the full row objects can be noticeable, .5 to 1.5 seconds for lists with over 10,000 items. Because of this, the programmer may not want the fix up to happen during the insertion or deletion of a column.

Manual update

With this method, after the column insert or delete, the programmer iterates through all the list-view items, removes the old invalid item data, creates a valid full row object, and sets the item data to the new full row object. The following code snippet shows the procedure:

```
list~deleteColumn(delCol)

do i = 0 to list~items - 1
  list~removeItemData(i)
  lvFullRow = list~getFullRow(i, .true)
  list~setItemData(i, lvFullRow)
end
```

15.123.2. Method Table

The following table lists the class and instance methods of the **`LvFullRow`** class:

Table 15.4. LvFullRow Class Method Reference

Method	Description
Class	**Methods**
fromArray	Instantiates a new **`LvFullRow`** object from an array of values. The string value of each item in the array is used as the text for the item and subitem(s).
new	Instantiates a new **`LvFullRow`** object.
Attribute	**Methods**
userData	The *userData* attribute allows the programmer to associate some data with a **`LvFullRow`** object.

Method	Description
Instance	**Methods**
addSubItem	Adds a subitem to this full row. Subitems are always added as the last subitem in the row.
insertSubItem	Inserts a new subitem into this full row and adjusts the subitem indexes for all existing subitems, when needed.
item	Returns the *LvItem* object of this full row.
removeSubItem	Removes the specified subitem from this full row.
subItem	Returns the subitem specified by *index* of this full row, or the **.nil** object if the subitem *index* is not valid.
subItems	Returns the count of subitems in this full row.

15.123.3. fromArray (Class Method)

```
>>--fromArray(--data--+-----------+--+------------+--+-----------------+--)--><
                      +-,-iIndex--+  +-,-strIfNil--+  +-,-useForSorting--+
```

Instantiates a new **LvFullRow** object from an array of values. The string value of each item in the array is used as the text for the item and subitem(s).

Arguments:

The arguments are:

data [required]

An array object. The string value of each item in the array is used as the text for the item and subitems of the list-view item. The item at index 1 of the array is used as the text for the list-view item. The item at index 2 is used as the text for the first subitem, the item at index 3 is used as the text for subitem 2, etc.. The array can not be sparse

iIndex [optional]

The item index to use for the full row. If omitted, 0 is used for the item index. See the remarks, depending on the use of the returned full row object, specifying an index is often not needed.

strIfNil [optional]

If the item at any index in *data* array is the **.nil** object, then the text for the item or subitem corresponding to that index will be the string value of the **.nil** object. Which is *the NIL object*. The *strIfNil* argument can be used to specify text to use instead of *the NIL object*.

useForSorting [optional]

This argument can be either true or false. Recall that the list-view control allows the programmer to *attach* an item data value to any or all of the list-view items. If the item data for every list-view item is a **LvFullRow** object, the ooDialog framework can perform internal *sorting* of the list-view items. Use true to specify that the **LvFullRow** object being instantiated should be set as the item data value for the list-view item the object represents. If false, the full row object is not set as the item data. The default is false.

Return value:

A newly instantiated **LvFullRow** object on success, the **.nil** object on error.

Remarks:

The returned full row object is intended to be used to insert an item into a list-view. Other uses of the full row object may cause unpredictable results. It is the responsibility of the programmer to determine if the full row object is suitable for some other use. The programmer should consider that only the text attribute of the full row item and its subitems is set.

Recall that if the full row object is used in either the *addFullRow* or *prependFullRow* methods, the item index of the full row object is ignored. For the *insertFullRow* method, if the item index is 0, the method will work fine, the full row will be inserted as the first item in the list-view. Therefore the optional *itemIndex* argument is never needed, and, depending on the use of the full row object, may be a waste of time.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example queries a ooSQLite database and then populates the list-view with the returned result set:

```
sql  = 'SELECT * FROM foods ORDER BY name;'
data = dbConn~exec(sql, .true)
dbConn~close

self~insertListViewColumns(list, data[1])
do i = 2 to data~items
    fRow = .LvFullRow~fromArray(data[i], , 'null', .true)
    list~addFullRow(fRow)
end
```

15.123.4. new (Class Method)

```
                +--------------+
                V              |
>>--new(--item--+--------------+--+----------+--)--------------><
                +--,--subItem--+  +-,-sorting-+
```

Instantiates a new **`LvFullRow`** object.

Arguments:

The arguments are:

item [required]

A *LvItem* object that specifies the attributes of the list-view item being added.

subItem [optional]

A *LvSubItem* object that specifies the attributes of a subitem of the list-view item being added. The argument can repeat has many times as needed to specify all the subitems of the item. The subitems are added to the item in the order of the arguments to the *new* method.

sorting [optional]

The very last argument can optionally be **`.true`** or **`.false`** to signal that the **`LvFullRow`** object itself is to be assigned as the item data value of the list-view items. This usage is explained in the remarks section.

Return value:

Returns the new **`LvFullRow`** object.

Remarks:

Each subitem argument must be specified. If a subitem is omitted and a subitem argument follows it, a syntax condition is raised. That is, if a subitem is specified, its preceding subitem must not have been omitted. These statements are acceptable:

```
      row = .LvFullRow~new(item)
      row = .LvFullRow~new(item, si1, si2)
      row = .LvFullRow~new(item, si1, si2, si3)
```

These statements will raise a syntax condition:

```
      row = .LvFullRow~new(item, , si2)
      row = .LvFullRow~new(item, si1, , si3)
```

In addition to specifying **`LvSubItem`** objects, the last argument can be either true or false. Recall that the list-view control allows the programmer to *attach* an item data value to any or all of the list-view items. If the item data for every list-view item is a **`LvFullRow`** object, the ooDialog framework can perform internal *sorting* of the list-view items. The last argument can be true to specify that the **`LvFullRow`** object being instantiated should be set as the item data value for the list-view item the **`LvFullRow`** object represents.

This feature is really only of use if the full row object is used to add or insert an item into the list-view. If the full row object is not to be used to add an item to the list-view, then there is no reason to use the last argument of true or false.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows a method that creates an array of list-view full rows from NFL football player data. The array is later used to insert the rows into a list-view control.

```
::method getPlayers private

  players = .NFLPlayer~playersFromDataSource("nflPlayers.txt")

  self~masterList = .MasterPlayerList~new(players)

  playerRows = .array~new(players~items)
  do i = 1 to players~items
    j = i - 1
    p = players[i]
    lvi = .LvItem~new(j, , p~name, , p)
    lvsi1 = .LvSubItem~new(j, 1, p~team)
    lvsi2 = .LvSubItem~new(j, 2, p~position)
    lvsi3 = .LvSubItem~new(j, 3, p~rating)

    r = .LvFullRow~new(lvi, lvsi1, lvsi2, lvsi3)
    playerRows[i] = r
  end

  return playerRows
```

15.123.5. userData (Attribute)

```
>>--userData-----------------------------------------------------><

>>--userData = varName-------------------------------------------><
```

The *userData* attribute allows the programmer to associate some data with a **`LvFullRow`** object. The data assigned can be any Rexx object the programmer desires.

userData get:
Returns the user data object assigned to this full row object, or the **`.nil`** object if no user data has been associated with this full row.

userData set:
Assigns a user data object to this full row object. There is no restriction on what type of object this may be.

Remarks:
The Windows list-view control allows an *data* value to be associated with any or all items in the list-view. This is convenient for a number of things. However, if the ooDialog program is using **`LvFullRow`** objects for *internal* sorting, the full row object is assigned as the data value for each item. This prevents the programmer from using the list-view's item data for some other purpose. The *userData* attribute is provided for these circumstances. The programmer assigns a value to the *userData* attribute of the full row object that needs to be associated with the list-view item. Since the full row object is assigned to the list-view item data, the value assigned to the *userData* attribute is in effect assigned to the list-view item.

Example:
This example constructs a full row object to be displayed as an item in a list-view. The data for each item comes from records in a database. In order to keep track of which record in the database corresponds to each list-view item, the record ID is stored in the *userData* attribute of the full row object. The value at index 1 of the rec array is the rowid of the record:

```
::method fullRowFromRecord unguarded
    use strict arg rec, index = 0

    lvItem = .LvItem~new(index, rec[2])
    lvSub1 = .LvSubItem~new(index, 1, rec[3])
    lvSub2 = .LvSubItem~new(index, 2, rec[4])
    lvSub3 = .LvSubItem~new(index, 3, rec[6])
    lvSub4 = .LvSubItem~new(index, 4, rec[7])
    lvSub5 = .LvSubItem~new(index, 5, rec[8])

    lvFullRow = .LvFullRow~new(lvItem, lvSub1, lvSub2, lvSub3, lvSub4, lvSub5, .true)
    lvFullRow~userData = rec[1]

    return lvFullRow
```

15.123.6. addSubItem

```
>>--addSubItem(--subitem--)----------------------><
```

Adds a subitem to this full row. Subitems are always added as the last subitem in the row.

Arguments:
The single argument is:

subitem [required]

A *LvSubItem* object that specifies the subitem being added.

Return value:

Returns the index of the added subitem on success, or 0 on error.

Remarks:

The *addSubItem* method allows the programmer to add subitems to the full row after the **`LvFullRow`** object has been instantiated. Typically, all subitems would be added to the full row at the time of instantiation through the *new* method. In addition to the *addSubItem* method, the *insertSubItem* method can be used to insert a subitem into the middle of the subitem list.

Details

Raises syntax errors when incorrect usage is detected.

15.123.7. insertSubItem

```
>>--insertSubItem(--subItem--,--colIndex--)------><
```

Inserts a new subitem into this full row and adjusts the subitem indexes for all existing subitems, when needed.

Arguments:

The arguments are:

subitem [required]

A *LvSubItem* object that specifies the subitem being inserted.

colIndex [required]

The one-based index that specifies which column the subitem is inserted at. The index can not be 0 and can not be greater than the current count of subitems + 1. That is, a subitem can not be inserted in a way that leaves the subitems non-contiguous.

Return value:

Returns true on success, false on error.

Remarks:

The *insertSubItem* method allows the programmer to add subitems to the full row after the **`LvFullRow`** object has been instantiated. Typically, all subitems would be added to the full row at the time of instantiation through the *new* method. In addition to the *insertSubItem* method, the *addSubItem* method can be used to add a subitem to the end of the subitem list.

Inserting a new column into the middle of the exsiting subitems, will of course make the subitem index in the subitems following the newly inserted subitem invalid. The ooDialog framework will automatically fix the subitem indexes where needed. In addition, both the *item* and *subItem* attributes in the newly inserted *subitem* object are set to the collect values automatically. Because of this, the *subitem* object argument does not necessarily have to have correct item and subitem index values.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example inserts a new subitem for column 3. Notice that when instantiating the **`LvSubItem`** object, the code does not bother with setting the *item* and *subitem* correctly. Rather it relies on the *insertSubItem* method to set them correctly:

```
    lvSubItem = .LvSubItem~new(1, 1, 'Apartment #13', 2)
    lvFull~insertSubItem(lvSubItem, 3)
```

15.123.8. item

```
>>--item-----------------------------------------><
```

Returns the *LvItem* object of this full row.

Arguments:

There are no arguments for this method.

Return value:

Returns the *LvItem* object for this full row. All full rows must have an item object so there is no way for this method to fail.

Remarks:

Additional comments.

Example:

This example gets the **`LvFullRow`** for the item at index 5 of the list-view. It then gets the **`LvItem`** object and prints its information out to the display

```
    lvfull = list~getFullRow(5)
    lvitem = lvFull~item
    self~printItem(lvitem)

/* Output might be for example:

Item columnFormats:     an Array
Item columns:           an Array
Item columns count:     0
Item groupID:           -2
Item imageIndex:        3
Item indent:            0
Item index:             5
Item itemData:          a LvFullRow
Item itemState:         FOCUSED SELECTED
Item itemStateMask:
Item mask:              GROUPID IMAGE PARAM TEXT
Item overlayImageIndex: 0
Item stateImageIndex:   0
Item text:              Clerk

*/
```

15.123.9. removeSubItem

```
>>--removeSubItem(--index--)---------------------><
```

Removes the specified subitem from this full row.

Arguments:

The arguments are:

index [required]

The index of the subitem to remove. This is the same as the column index for the subitem. *index* must be a valid, existing, subitem index. This means it can not be 0, or greater than the count of current subitems.

Return value:

Returns the removed **`LvSubItem`** object on success, or the **`.nil`** object on error.

Remarks:

The *removeSubItem* method allows the programmer to remove subitem from the full row after the **`LvFullRow`** object has been instantiated. Typically, all subitems would be set correctly in the full row at the time of instantiation through the *new* method.

Removing a column from the middle of the exsiting subitems, will of course make the subitem index in the subitems following the removed subitem invalid. The ooDialog framework will automatically fix the subitem indexes where needed.

Details

Raises syntax errors when incorrect usage is detected.

15.123.10. subItem

```
>>--subItem(--index--)---------------------------><
```

Returns the subitem specified by *index* of this full row, or the **`.nil`** object if the subitem *index* is not valid.

Arguments:

The single argument is:

index [required]

The one-based index of the subitem requested.

Return value:

Returns the requested *LvSubItem* object on success, or the **`.nil`** object on error.

Example:

This example gets the subitem in column 3 of the item at index 4 in the list-view and prints out its information:

```
  lvfull  = list~getFullRow(4)
  subItem = lvFull~subItem(3)
  self~printSubItem(subItem)

/* Output might be:

Subitem imageIndex:        11
Subitem item:              4
Subitem mask:              IMAGE TEXT
Subitem subitem:           3
Subitem text:              ca@sharp.org
```

```
*/
```

15.123.11. subItems

```
>>--subItems-------------------------------------><
```

Returns the count of subitems in this full row.

Arguments:

There are no arguments for this method.

Return value:

Returns the count of subitems in this full row.

Example:

This example ...

```
  lvfull  = list~getFullRow(4)
  if lvfull~subitems == 0 then do
      say 'No subitems for row at index 4'
      return
  end

  do i = 1 to lvfull~subitems
      subItem = lvFull~subItem(i)
      self~printSubItem(subItem)
  end

/* Output might be:

Item imageIndex:        13
Item item:              4
Item mask:              IMAGE TEXT
Item subitem:           1
Item text:              Cienna

Item imageIndex:        18
Item item:              4
Item mask:              IMAGE TEXT
Item subitem:           2
Item text:              Acer

Item imageIndex:        11
Item item:              4
Item mask:              IMAGE TEXT
Item subitem:           3
Item text:              ca@sharp.org

*/
```

15.124. LvItem Class

A **`LvItem`** object represents a single item in a list-view control. List-view items can have a label, an icon, a current state, an application-defined value, a group ID, a set of columns to display in Tile view, column format definitions, a state icon, and an overlay icon. The attributes of the **`LvItem`** object reflect the values of the label, icon, state, etc., for the list-view item.

`LvItem` objects can be used to set or modify the values of a list-view item, or to receive information about an item. In addition **`LvItem`** objects are used in *LvFullRow* objects. Each **`LvFullRow`** object is composed of at least one **`LvItem`** object.

Note that ooDialog does not currently support some of the newer features in list-views. The **`LvItem`** class is part of the first steps to enhance ooDialog to support all the list-view features. The **`LvItem`** class has support for attributes that will become important in an enhanced *ListView* class, but are not currently relevant. Support for list-view groups, Tile view, and columns, is part of a planned enhancement.

15.124.1. Method Table

The following table lists the class and instance methods of the **`LvItem`** class:

Table 15.5. LvItem Class Method Reference

Method	Description
Class Methods	**Class Methods**
new	Instantiates a new **`LvItem`** object.
Attribute Methods	**Attribute Methods**
columnFormats	An array of column format values specifying the format of each item in a column in Tile view.
columns	An array of column indexes specifying which columns are displayed for this item in Tile view, and the order of those columns.
groupID	Reflects the group ID of this item.
imageIndex	Reflects the index in the icon *image* lists for this item.
indent	Reflects the number of image widths to indent this item.
index	Reflects the index of this item in the list-view.
itemData	Reflects the item data value for this item.
itemState	Reflects this item's state.
itemStateMask	The *itemStateMask* attribute specifies which states in the *itemState* attribute are valid.
mask	The *mask* attribute is used to specify which values of the **`LvItem`** object are valid to set or receive information of the list-view item.
overlayImageIndex	Reflects the one-based index of the overlay image for this item.
stateImageIndex	Reflects the one-based index of the state image for this item.
text	Reflects the text, the label, for this item.

15.124.2. new (Class Method)

```
>>--new(--+--------+--+-------+--+--------------+--+------------+--+-------+---->
          +--indx--+  +-,-txt-+  +-,-imageIndex-+  +-,-itemData-+  +-,-msk-+

>---+-------------+-+----------------+-+----------+-+-----------+-+--------+--)--><
    +-,-itemState-+ +-,-itemStateMsk-+ +-,-indent-+ +-,-groupID-+ +-,-cols-+
```

Instantiates a new **`LvItem`** object. **`LvItem`** objects can be used in some *Listview* methods such as *getItem*. **`LvItem`** objects are used in *LvFullRow* . It is anticipated that future enhancements of ooDialog will make more use of **`LvItem`** objects.

Some of the uses for **`LvItem`** objects are to set or retrieve some, or all, of the information the underlying list-view control maintains for each list-view item.

Arguments:

The arguments are:

indx [optional]

The zero-based index of this item. The argument can not be less than 0. If this arugment is omitted it defaults to 0. The *indx* argument sets the *index* attribute of this list-view item.

Note that depending on the use of this list-view item, the ooDialog framework may modify the index attribute. For instance if the **`LvItem`** is contained in a *LvFullRow* object and the row is added to a list-view, the ooDialog framework will adjust the *index* attribute to reflect the actual index of the added item.

txt [optional]

Sets the text, or label, for this list-view item. The *txt* argument sets the *text* attribute of this list-view item.

imageIndex [optional]

The zero-based index of the item's icon in the control's *image* list. For any item there is only 1 image index, so the index must be the same in both the large and small image lists. The *imageIndex* argument can also be one of the following list-view *constants*. If this argument is omitted, the default is IMAGENONE:

IMAGECALLBACK	IMAGENONE

IMAGECALLBACK

Indicates the list-view control should send the GETDISPINFO notification message to retrieve the icon index for the item.

IMAGENONE

The item does not have an icon. This is the default if this argument omitted.

The *imageIndex* argument sets the *imageIndex* attribute of this list-view item.

itemData[optional]

An item data value for this item. The list-view control allows the programmer to *assign* an item data value to any, or all, list-view items. The item data value can be any Rexx object.

The *itemData* argument sets the *itemData* attribute of this list-view item.

msk [optional]

A list of blank separate keywords that specify which attributes of this **`LvItem`** object contain data to be set or which attributes are being requested. In general, the ooDialog framework will try to set the *mask* attribute correctly without the programmer having to worry about this argument. As each attribute of the **`LvItem`** object is set, the appropriate keyword is added to the *mask* attribute. However, if this object is going to be used to retrieve information about the underlying list-view item, the ooDialog framwork has no way to know what attributes the programmer wishes to retrieve. For this use of the **`LvItem`** object, the programmer should set the mask to the value that will retrieve the information she wants.

The *msk* argument can have zero or more of the following keywords, case is not significant:

ALL	IMAGE	STATE
COLFMT	INDENT	TEXT
COLUMNS	NORECOMPUTE	
GROUPID	PARAM	

ALL

A convenience keyword, the same as specifying a string with every keyword in it.

COLFMT

Requires Windows Vista or later. The *columnFormats* attribute is valid or should be set.

COLUMNS

Requires Common Control *Library* version 6.0 or later. The *columns* attribute is valid or should be set.

GROUPID

Requires Common Control *Library* version 6.0 or later. The *groupID* attribute is valid or should be set.

IMAGE

The *imageIndex* attribute is valid or should be set.

INDENT

The *indent* attribute is valid or should be set.

NORECOMPUTE

Used when the **`LvItem`** object will be used to retrieve information through the *getItem* method. Signals the list-view control not to generate a GETDISPINFO notification to retrieve the text. Rather, the *text* attribute will contain *lpStrTextCallBack*.

PARAM

The *itemData* attribute is valid or should be set.

STATE

The *itemState* attribute is valid or should be set.

TEXT

The *text* attribute is valid or should be set.

The *msk* argument sets the *mask* attribute of this list-view item.

itemState [optional]

A list of blank separate keywords that specify the state for the list-view item. An item's state determines its appearance and functionality. The list can contain zero or more of the following keywords:

CUT	FOCUSED
DROPHILITED	SELECTED

CUT

The list-view item is marked for a cut-and-paste operation

DROPHILITED

The list-view item is highlighted as a drag-and-drop target.

FOCUSED

The list-view item has the focus, so it is surrounded by a standard focus rectangle. Although more than one item may be selected, only one item can have the focus.

SELECTED

The list-view item is selected. The appearance of a selected item depends on whether it has the focus and also on the system colors used for selection.

Sets the *itemState* attribute of this list-view item.

itemStateMsk [optional]

Sets the *itemStateMask* attribute of this list-view item.

A string of blank separated keywords specifying which state values to retrieve or set. For example, if only the keyword SELECTED is used, only the value of the SELECTED state will be retrieved in the *state* attribute. This also allows the programmer to modify one or more state values without first retrieving the current state values. For instance, setting the item state mask to SELECTED and setting the item state attribute to the empty string will clear the SELECTED state without changing any of the other state value, if the **`LvItem`** object is used to modify a list-view item. If the *itemStateMask* is set to ALL and the *itemState* is set to the empty sting will clear the SELECTED, CUT, FOCUSED and HILITED states, if the **`LvItem`** object is used to modify a list-view item.

The acceptable keywords are the same as those for the *itemState* argument, with the addition of the ALL keyword:

ALL	DROPHILITED	SELECTED
CUT	FOCUSED	

ALL

All keywords in the *itemState* attribute are valid.

CUT

The CUT keyword is valid in the *itemState* attribute.

DROPHILITED

The DROPHILITED keyword is valid in the *itemState* attribute.

FOCUSED

The FOCUSED keyword is valid in the *itemState* attribute.

SELECTED

The SELECTED keyword is valid in the *itemState* attribute.

indent [optional]

The number of image widths to indent the item. A single indentation equals the width of an item image. Therefore, the value 1 indents the item by the width of one image, the value 2 indents by two images, and so on. If this argument is omitted, the indent is set to 0. The *indent*argument sets the *indent* attribute of this list-view item.

groupID [optional]

Requires Common Control *Library* version 6.0 or later.

The group ID this item belongs to, or one of the following **`LvItem`** constants. If this argument is omitted, the default is the GROUPIDNONE constant:

GROUPIDCALLBACK	GROUPIDNONE

GROUPIDCALLBACK

Indicates the list-view control should send the GETDISPINFO notification message to retrieve the group index of the item.

GROUPIDNONE

The item does not belong to a group. If you omit this argument, this is the default

The *groupID* argument sets the *groupID* attribute of this list-view item.

cols [optional]

Requires Common Control *Library* version 6.0 or later.

The *cols* argument must be an array of subitem indexes. The indexes in the array specify which columns are displayed for this item, in *Tile* view, and the order of those columns. The array can not be sparse. The *cols* argument sets the *columns* attribute of this list-view item.

Return value:

Returns a newly instantiated **LvItem** object.

Remarks:

Note that not all of the **LvItem** object's attributes can be set through the *new* method. The *overlayImageIndex*, *stateImageIndex*, and *columnFormats* attributes can be set through their attribute methods after the **LvItem** object has been instantiated.

How the new **LvItem** object should be instantiated depends on how the object will be used. Typically, if the object is going to be used to retrieve information about a list-view item, only the *indx* and *msk* arguments would be used in the *new* method. The *indx* argument would be set to the index of the item and the *msk* argument would be set for all the information that was wanted to be retrieved.

On the other hand, if the **LvItem** is going to be used to set the information for a list-view item, the typical process would be to set the index for the item using the *indx* argument and set the values for all the attributes needed to define the item. In this case, the *msk* argument can usually be omitted. For each attribute of the **LvItem** that is assigned a value, the proper keyword is added to the mask by the ooDialog framework.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example instantiates a **LvItem** object that will be used in a *LvFullRow* object to insert an item into a list-view. Only the label of the item and the image index are needed for the purposes of the application, so the other attributes of the **LvItem** are not assigned values. The ooDialog framework will automatically set the proper mask value:

```
lvItem = .LvItem~new(2, "Mechanical Engineer", 0)
...
lvFullRow = .LvFullRow~new(lvItem, lvSub1, lvSub2, lvSub3, .true)
list~addFullRow(lvFullRow)
```

This example instantiates a **LvItem** object that will be used to query the information maintained by the list-view for the item at index 21. All information for the item is desired, so the masks are set to receive all information. Note that the value of the list-view *constants* IMAGENONE and GROUPIDNONE are both -2:

```
lvItem = .LvItem~new(21, , , , 'ALL', , 'ALL')
if list~getItem(lvItem) then do
```

```
        self~printItem(lvItem)
    end
    else do
        -- some unexpected error ...
        say 'Error'
    end

::method printItem
    use strict arg lvItem

    say 'Item columnFormats:    ' lvItem~columnFormats
    say 'Item columns:          ' lvItem~columns
    say 'Item columns count:    ' lvItem~columns~items
    say 'Item groupID:          ' lvItem~groupID
    say 'Item imageIndex:       ' lvItem~imageIndex
    say 'Item indent:           ' lvItem~indent
    say 'Item index:            ' lvItem~index
    say 'Item itemData:         ' lvItem~itemData
    say 'Item itemState:        ' lvItem~itemState
    say 'Item itemStateMask:    ' lvItem~itemStateMask
    say 'Item mask:             ' lvItem~mask
    say 'Item overlayImageIndex:' lvItem~overlayImageIndex
    say 'Item stateImageIndex:  ' lvItem~stateImageIndex
    say 'Item text:             ' lvItem~text
    say




/* Output might be:

Item columnFormats:     an Array
Item columns:           an Array
Item columns count:     0
Item groupID:           -2
Item imageIndex:        -2
Item indent:            0
Item index:             21
Item itemData:          a LvFullRow
Item itemState:
Item itemStateMask:     FOCUSED SELECTED CUT DROPHILITED
Item mask:              COLFMT COLUMNS GROUPID IMAGE INDENT NORECOMPUTE PARAM STATE TEXT
Item overlayImageIndex: 0
Item stateImageIndex:   1
Item text:              Line 22

*/
```

15.124.3. columnFormats (Attribute)

```
>>--columnFormats--------------------------------><

>>--columnFormats = varName----------------------><
```

An array of column format values specifying the format of each item in a column in Tile view. See the remarks.

columnFormats get:

Returns an array of column formats. Each index of the array specifies the format of the corresponding column in Tile view. The formats are specified by one of the LVCFVT_* *constants* of the **`ListView`** class.

columnFormats set:

To specify the format for each column in Tile view assign an array to this attribute. The number of items in the array must be the same as the number of items in the array assigned to the *columns* attribute. Each index of the array should contain a valid column format. Specify the format using one of the LVCFVT_* *constants* of the **`ListView`** class.

Remarks:

In Tile view, the item label is displayed to the right of the icon. The programmer can specify additional subitems (corresponding to columns in Details view), to be displayed on lines below the item label. The *columns* attribute must be an array that contains the indexes of the subitems to be displayed.

The *columnFormats* attribute specifies the format for each subitem in the *columns* array. The *columnFormats* attribute does not have to be set if the *columns* attribute is set. In this case the default format for each column is used. But, if the *columnFormats* attribute is set, the *columns* attribute must also be set and the two arrays must be the same size.

For each index in the *columns* array, the same index in the *columnFormats* array specifies the format for that column. The *columnFormats* attribute has no effect in standard Tile views. For extended Tile views, the LVCFMT_LINE_BREAK, LVCFMT_FILL, LVCFMT_WRAP, LVCFMT_NO_TITLE, and LVCFMT_TILE_PLACEMENTMASK formats are valid.

ooDialog does not currently support Tile view, but it is likely that a future enhancement will. The *columnFormats* attribute is part of the first steps towards implementing the support for Tile view.

Details

Requires Windows Vista or later.

Raises syntax errors when incorrect usage is detected.

15.124.4. columns (Attribute)

```
>>--columns---------------------------------------><

>>--columns = varName-----------------------------><
```

An array of column indexes specifying which columns are displayed for this item in Tile view, and the order of those columns.

columns get:

Returns an array of column indexes. Each index in the array specifies the index of a column to be displayed in tile view. The order of the column indexes in the array specifies the order the columns are displayed in.

columns set:

To set which columns are displayed in Tile view, assign an array to this attribute. The number of items in the array specifies how many columns are displayed. The maximum is 20. The array can not be sparse. Each index of the array should contain the subitem index to be displayed. The subitems are displayed in the same order as their index appears in the array.

Remarks:

In Tile view, the item label is displayed to the right of the icon. The programmer can specify additional subitems (corresponding to columns in Details view), to be displayed on lines below the item label. The *columns* attribute must be assigned an array that contains the indexes of the subitems to be displayed.

The *columnFormats* attribute can be used to specify the format for each subitem in the *columns* array. The *columnFormats* attribute does not have to be set if the *columns* attribute is set. In this case the default format for each column is used. But, if the *columnFormats* attribute is set, the *columns* attribute must also be set and the two arrays must be the same size. For each index in the *columns* array, the same index in the *columnFormats* array specifies the format for that column.

ooDialog does not currently support Tile view, but it is likely that a future enhancement will. The *columnFormats* attribute is part of the first steps towards implementing the support for Tile view.

Details

Requires Common Control *Library* version 6.0 or later.

Raises syntax errors when incorrect usage is detected.

15.124.5. groupID (Attribute)

```
>>--groupID--------------------------------------><

>>--groupID = varName----------------------------><
```

Reflects the group ID of this item.

groupID get:

Returns the group ID of this item. The ID can also be one of the group ID *constants* supplied by the **ListView** class, GROUPIDNONE or GROUPIDCALLBACK.

groupID set:

Set this attribute to the value of the identifier of the group that this item belongs to, or to one of the group ID *constants* supplied by the **ListView** class, either GROUPIDNONE or GROUPIDCALLBACK. Do **not** set this attribute to 0.

Remarks:

Setting the *groupID* attribute to 0 will cause some uses of the **LvItem** to fail unless the list-view has groups enabled. The best strategy to avoid failures is to new set the attribute to 0.

ooDialog does not currently support list-view groups, but it is likely that a future enhancement will add this support. The *groupID* attribute is part of the first steps towards implementing the support for list-view groups.

Details

Raises syntax errors when incorrect usage is detected.

15.124.6. imageIndex (Attribute)

```
>>--imageIndex-----------------------------------><

>>--imageIndex = varName-------------------------><
```

Reflects the index in the icon *image* lists for this item.

imageIndex get:

Returns this item's image index, or one of the image *constants* supplied by the **`ListView`** class, IMAGENONE or IMAGECALLBACK.

imageIndex set:

Set this attribute to the index in the icon image lists for this item, or to one of the image *constants* supplied by the **`ListView`** class, IMAGENONE or IMAGECALLBACK.

Remarks:

In general, if the programmer does not explicitly set this attribute, the ooDialog framework will set it to the IMAGENONE constant. When the **`LvItem`** object is used to retrieve information about an item, the list-view control will return the IMAGECALLBACK value unless the image index has been explicitly set by the application.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example will set, or change, the image index of the item at index 13 to 3. None of the other list-view item information will be changed.

```
lvItem = .LvItem~new(13, , 3)
list~modifyItem(lvItem)
```

15.124.7. indent (Attribute)

```
>>--indent---------------------------------------><

>>--indent = varName-----------------------------><
```

Reflects the number of image widths to indent this item.

indent get:

Returns the indent value for this item

indent set:

Set this attribute to a non-negative whole number to set the indent for this item

Remarks:

A single indentation equals the width of an item image. Therefore, the value 1 indents the item by the width of one image, the value 2 indents by two images, and so on.

Details

Raises syntax errors when incorrect usage is detected.

15.124.8. index (Attribute)

```
>>--index----------------------------------------><

>>--index = varName------------------------------><
```

Reflects the index of this item in the list-view.

index get:
Returns the zero-based index of this item.

index set:
Set this attribute to a zero-based index to specify which item in the list-view this item represents.

Details
Raises syntax errors when incorrect usage is detected.

15.124.9. itemData (Attribute)

```
>>--itemData-------------------------------------><

>>--itemData = varName---------------------------><
```

Reflects the item data value for this item.

itemData get:
Returns the item data value of this item, or the **`.nil`** object if no item data has been set for this item.

itemData set:
Set this attribute to any value the programmer wants to associate with this item. The list-view control allows the application, (the programmer) to assign a user value to any list-view item. The *setItemData* method can also be used to set the item data for an item. The value can be any Rexx object that the programmer wants to use.

Details
Raises syntax errors when incorrect usage is detected.

Example:
This example retrieves the current item data for list-view item 16:

```
    lvItem = .LvItem~new(16, , , , 'PARAM')
    list~getItem(lvItem)
    say 'The item data for item 16 is:' lvItem~itemData

/* Output might be:

  The item data for item 16 is: a LvFullRow

*/
```

15.124.10. itemState (Attribute)

```
>>--itemState------------------------------------><

>>--itemState = varName--------------------------><
```

Reflects this item's state.

itemState get:

Returns a blank separated list of keywords that indicate the item's state. The possible keywords are listed in the remarks.

itemState set:

To set this item's state, assign a string containing a list of state keywords. The possible keywords are listed in the remarks.

Remarks:

The state string contains a list of blank separated keywords. When assigning the string to this attribute case is insignificant. When getting the value of this attribute, the returned keywords will be all upper-cased. The possible keywords are:

CUT	FOCUSED
DROPHILITED	SELECTED

CUT

The list-view item is marked for a cut-and-paste operation

DROPHILITED

The list-view item is highlighted as a drag-and-drop target.

FOCUSED

The list-view item has the focus, so it is surrounded by a standard focus rectangle. Although more than one item may be selected, only one item can have the focus.

SELECTED

The list-view item is selected. The appearance of a selected item depends on whether it has the focus and also on the system colors used for selection.

Note that the itemStateMask attribute needs to be used in conjunction with this attribute. The *itemStateMask* specifies which states in the *itemState* are valid. If a state, say SELECTED, is specified in the *itemStateMask*, but not in the *itemState*, that state is turned off. If a state is specified in both the *itemState* and *itemStateMask* attributes, that state is turned on.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example sets the state of the item at index 1 to selected and focused:

```
lvItem = .LvItem~new(1)
lvItem~itemState = "SELECTED FOCUSED"
lvItem~itemStateMask = "SELECTED FOCUSED"
list~modifyItem(lvItem)
```

15.124.11. itemStateMask (Attribute)

```
>>--itemStateMask--------------------------------><

>>--itemStateMask = varName----------------------><
```

The *itemStateMask* attribute specifies which states in the itemState attribute are valid.

itemStateMask get:

Returns a blank separated list of state keywords that indicate which states in the *itemState* attribute are valid. The possible keywords are listed in the remarks

itemStateMask set:

To specify the valid state keywords in the *itemState* attribute, assign a list of blank separated keywords to this attribute. The possible keywords are listed in the remarks.

Remarks:

The state mask string contains a list of blank separated keywords. When assigning the string to this attribute case is insignificant. When getting the value of this attribute, the returned keywords will be all upper-cased. The possible keywords are:

ALL	DROPHILITED	SELECTED
CUT	FOCUSED	

ALL

All keywords in the *itemState* attribute are valid.

CUT

The CUT keyword, or lack of, is valid in the *itemState* attribute.

DROPHILITED

The DROPHILITED keyword, or lack of, is valid in the *itemState* attribute.

FOCUSED

The FOCUSED keyword, or lack of, is valid in the *itemState* attribute.

SELECTED

The SELECTED keyword, or lack of, is valid in the *itemState* attribute.

Note that the *itemState* attribute is used in conjunction with this attribute. The *itemStateMask* specifies which states in the *itemState* are valid. If a state, say SELECTED, is specified in the *itemStateMask*, but not in the *itemState*, that state is turned off. If a state is specified in both the *itemState* and *itemStateMask* attributes, that state is turned on.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example uses the *itemStateMask* to turn off the FOCUSED state of the item without changing any of the other states of the item:

```
lvItem = .LvItem~new(10)
lvItem~itemStateMask = "FOCUSED"
list~modifyItem(lvItem)
```

15.124.12. mask (Attribute)

```
>>--mask-----------------------------------------><

>>--mask = varName-------------------------------><
```

The *mask* attribute is used to specify which values of the **`LvItem`** object are valid to set or receive the information of the list-view item.

mask get:

Returns a blank separated list of keywords specifying which attributes of this item are valid. The possible keywords are listed in the remarks section.

mask set:

To specify which attributes in this item are to be used to set or receive information, assign a list of blank separated keywords to this attribute. The possible keywords are listed in the remarks section.

Remarks:

The *mask* attributed contains a list of blank separate keywords that specify which attributes of this **`LvItem`** object contain data to be set or which attributes are being requested. In general, when setting data, the ooDialog framework will try to set the *mask* attribute correctly without the programmer having to worry about this argument. As each attribute of the **`LvItem`** object is set, the appropriate keyword is added to the *mask* attribute.

However, if this object is going to be used to retrieve information about the underlying list-view item, the ooDialog framwork has no way to know what attributes the programmer wishes to retrieve. For this use of the **`LvItem`** object, the programmer should set the mask to the value that will retrieve the information he wants.

The *mask* attribute can have zero or more of the following keywords. When assigning the string to this attribute case is insignificant. When getting the value of this attribute, the returned keywords will be all upper-cased:

ALL	IMAGE	STATE
COLFMT	INDENT	TEXT
COLUMNS	NORECOMPUTE	
GROUPID	PARAM	

ALL

A convenience keyword, the same as specifying a string with every keyword in it.

COLFMT

Requires Windows Vista or later. The *columnFormats* attribute is valid or should be set.

COLUMNS

Requires Common Control *Library* version 6.0 or later. The *columns* attribute is valid or should be set.

GROUPID

Requires Common Control *Library* version 6.0 or later. The *groupID* attribute is valid or should be set.

IMAGE

The *imageIndex* attribute is valid or should be set.

INDENT

The *indent* attribute is valid or should be set.

NORECOMPUTE

Used when the **`LvItem`** object will be used to retrieve information through the *getItem* method. Signals the list-view control not to generate a GETDISPINFO notification to retrieve the text. Rather, the *text* attribute will contain *lpStrTextCallBack*.

PARAM

The *itemData* attribute is valid or should be set.

STATE

The *itemState* attribute is valid or should be set.

TEXT

The *text* attribute is valid or should be set.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example sets the text label, the icon index, the item data, and the indent for the item at index 11, without changing any other information of the item. In this example, the mask attribute is set explicitly to make a point. But it is not actually needed:

```
lvItem = .LvItem~new(11, 'Acme Clocks', 2, contactInfo, "TEXT IMAGE PARAM INDENT", , ,
2)

list~modifyItem(lvItem)
```

This code snippet will work exactly the same as the example above:

```
lvItem = .LvItem~new(11, 'Acme Clocks', 2, contactInfo, , , , 2)

list~modifyItem(lvItem)
```

15.124.13. overlayImageIndex (Attribute)

```
>>--overlayImageIndex----------------------------><

>>--overlayImageIndex = varName------------------><
```

Reflects the one-based index of the overlay image for this item.

overlayImageIndex get:

Returns the one-based index of the overlay image for this item. 0 indicates the item has no overlay image.

overlayImageIndex set:

To specify an overlay image for this item assign the one-based index of the overlay image in the image list. Assign 0 to indicate the item should not use an overlay image. Valid indexes are 0 through 15. This is a limitation imposed by the operating system.

Remarks:

Both the full-sized icon image list and the small icon image list can have overlay images. The overlay image is superimposed over the item's icon image. If the overlay index is zero, the item has no overlay image.

The overlay image index is actually part of the state value for an item in the underlying list-view control. The ooDialog framework hides some of the complexity of separating the overlay index from the state value by making the index a separate attribute in the **`LvItem`** object. If the programmer wants to retrieve the overlay index only, it is necessary to set the STATE flag in the *mask* attribute.

Currently, ooDialog does not have support for overlay images in the *ImageList* class. However, it is anticipated that support may be added in a future enhancement. Support for overlay image indexes in the **`LvItem`** class is a first step towards implementing support for overlay images.

Details

Raises syntax errors when incorrect usage is detected.

15.124.14. stateImageIndex (Attribute)

```
>>--stateImageIndex------------------------------><

>>--stateImageIndex = varName--------------------><
```

Reflects the one-based index of the state image for this item.

stateImageIndex get:

Returns the one-based index of the state image for this item. 0 indicates the item has no state image.

stateImageIndex set:

To specify a state image for this item assign the one-based index of the state image in the state image list. Assign 0 to indicate the item should not use an state image. Valid indexes are 0 through 15. This is a limitation imposed by the operating system.

Remarks:

The state image is displayed next to an item's icon to indicate an application-defined state. If the state image index is zero, the item has no state image.

The state image index is actually part of the state value for an item in the underlying list-view control. The ooDialog framework hides some of the complexity of separating the state index from the state value by making the index a separate attribute in the **`LvItem`** object. If the programmer wants to retrieve the state image index only, it is necessary to set the STATE flag in the *mask* attribute.

Currently, ooDialog does not have support for setting the state image list in the *setImageList* method of the **`ListView`** class. However, it is anticipated that support may be added in a future enhancement. Support for state image indexes in the **`LvItem`** class is a first step towards implementing support for state images.

Details

Raises syntax errors when incorrect usage is detected.

15.124.15. text (Attribute)

```
>>--text-----------------------------------------><
```

```
>>--text = varName-------------------------------><
```

Reflects the text, the label, for this item.

text get:

Returns the text for this item. If no text has been set for this item, the **`.nil`** object is returned. Note that setting the text to the empty string is different than not having the text set at all. In addition, the operarating system uses a special value to indicate to the list-view that it should generate a GETDISPINFO notification to retrieve the text for the item from the application. If this value is set, the string *lpStrTextCallBack* is returned as the text for this item.

text set:

Assign a string to this attribute to set the text for the item. The string must be less than or equal to 260 characters in length. In addition the special string, *lpStrTextCallBack*, can be used to set the text value for the item to the value that indicates the list-view should generate a GETDISPINFO notification. Case is not significant when using *lpStrTextCallBack*.

Remarks:

Currently, there is no support for the GETDISPINFO event notification in *connectListViewEvent* method. It is anticipated that support for this event will be added in the future. Until then, using the *lpStrTextCallBack* value will simply result in the item appearing with no text.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example gets the current text for an item, appends a string to the text, and sets the text for the item to the new value:

```
::method addQualifier private
    expose list
    use strict arg itemIndex, additionalText

    lvItem = .LvItem~new(itemIndex, , , , 'TEXT')
    if list~getItem(lvItem) then do
        lvItem~text = lvItem~text additionalText

        list~modifyItem(lvItem)
        return .true
    end
    else do
        return .false
    end
```

15.125. LvSubItem Class

A LvSubItem object represents a specific subitem of a single item within a list-view control. Each item in a list-view can have one or more subitems. Each subitem can have a text string and an icon. In report view, the text, and optionally the icon, is displayed in a column separate from the item's icon and label. To display the subitem's icon, the list-view must have the SUBITEMIMAGES *extended* style.

All items in a list-view control have the same number of subitems. The number of subitems is determined by the number of columns in the list-view control. When a column is *added* to a list-view control, its associated subitem index is specified. When a column is added to a list-view, the list-view control creates a subitem for every existing item with a default value of no text and no icon. When a column is removed from a list-view, the list-view deletes the information it maintains for every subitem of that column.

The list-view does not have to be in report view to have subitems. It does not have to be in report view to have columns added or deleted, the list-view can be in any view mode.

`LvSubItem` objects can be used to set the information for a subitem or to receive information concerning a subitem. In addition, *LvFullRow* objects can be composed of subitem objects in addition to a *LvItem* object.

15.125.1. Method Table

The following table lists the class and instance methods of the **`LvSubItem`** class:

Table 15.6. LvSubItem Class Method Reference

Method	Description
Class Methods	**Class Methods**
new	Instantiates a new **`LvSubItem`** object.
Attribute Methods	**Attribute Methods**
imageIndex	Reflects the index in the small icon *image* list for this subitem.
item	Reflects the index of the item in the list-view that this subitem belongs to.
mask	The *mask* attribute is used to specify which values of the **`LvSubItem`** object are valid to set or receive the information for a list-view item's subitem.
subItem	Reflects the subitem index of this **`LvSubItem`** object.
text	Reflects the text, the label, for this subitem.

15.125.2. new (Class Method)

```
>>--new(--item-,-subItem--+---------+--+-------------+--+--------+--)---------><
                          +-,-text--+  +-,-imageIndex-+  +-,-mask-+
```

Instantiates a new **`LvSubItem`** object with the parameters specified.

Arguments:

Instantiates a new **`LvSubItem`** object.

item [required]

The zero-based index of the item this subitem belongs to. The argument can not be less than 0. The *item* argument sets the *item* attribute of this subitem.

Note that depending on the use of this subitem, the ooDialog framework may modify the index attribute. For instance if the **`LvSubItem`** is contained in a *LvFullRow* object and the row is added to a list-view, the ooDialog framework will adjust the *index* attribute to reflect the actual index of the added item.

subItem [required]

The one-based index of this subitem. The argument can not be less than 1. The *subItem* argument sets the *subItem* attribute of this subitem.

Note that depending on the use of this subitem, the ooDialog framework may modify the *subItem* attribute. For instance if the **`LvSubItem`** is placed into a *LvFullRow* object, the ooDialog framework will adjust the *subItem* attribute to reflect the actual index of the column of the subitem in the full row.

text [optional]

Sets the text, or label, for this list-view subitem. The *text* argument sets the *text* attribute of this list-view subitem.

imageIndex [optional]

The zero-based index of the subitem's icon in the control's small icon *image* list. Note that the image index has no meaning if the list-view does not have the SUBITEMIMAGES *extended* style.

The *imageIndex* argument can also be one of the following list-view *constants*. If this argument is omitted, the default is IMAGENONE:

IMAGECALLBACK	IMAGENONE

IMAGECALLBACK

Indicates the list-view control should send the GETDISPINFO notification message to retrieve the icon index for the subitem.

IMAGENONE

The subitem does not have an icon. This is the default if this argument omitted.

The *imageIndex* argument sets the *imageIndex* attribute of this subitem.

mask [optional]

A list of blank separate keywords that specify which attributes of this **`LvSubItem`** object contain data to be set or which attributes are being requested. In general, the ooDialog framework will try to set the *mask* attribute correctly without the programmer having to worry about this argument. As each attribute of the **`LvSubItem`** object is set, the appropriate keyword is added to the *mask* attribute. However, if this object is going to be used to retrieve information about the underlying list-view subitem, the ooDialog framwork has no way to know what attributes the programmer wishes to retrieve. For this use of the **`LvSubItem`** object, the programmer should set the mask to the value that will retrieve the information she wants.

The *mask* argument can have zero or more of the following keywords, case is not significant:

ALL	NORECOMPUTE
IMAGE	TEXT

ALL

A convenience keyword, the same as using the string: *IMAGE NORECOMPUTE TEXT*.

IMAGE

The *imageIndex* attribute is valid or should be set.

NORECOMPUTE

Used when the **`LvSubItem`** object will be used to retrieve information through the *getSubitem* method. Signals the list-view control should not generate a GETDISPINFO notification to retrieve the text. Rather, the *text* attribute will contain *lpStrTextCallBack*.

TEXT

The *text* attribute is valid or should be set.

The *mask* argument sets the *mask* attribute of this subitem.

Return value:

Returns a newly instantiated and initialized **`LvSubItem`** object.

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example shows **`LvSubItem`** objects being instantiated and used in a *LvFullRow* object, which in turn is used to add a new item to the list-view. Note that the list-view has the extended SUBITEMIMAGES style and the use of the *imageIndex* argument in the *new* method. Each column in the list-view will have its own icon:

```
::method populateList private
    use strict arg list

    list~InsertColumnPx(0, "Title", 150)
    ret = list~InsertColumnPx(1, "Name", 75)
    ret = list~InsertColumnPx(2, "Last", 100)
    ret = list~InsertColumnPx(3, "e-mail", 150)

    list~addExtendedStyle("FULLROWSELECT UNDERLINEHOT ONECLICKACTIVATE SUBITEMIMAGES")

    lvItem = .LvItem~new(0, "Business manager", 6)
    lvSub1 = .LvSubItem~new(0, 1, "Tom", 14)
    lvSub2  = .LvSubItem~new(0, 2, "Sawyer", 26)
    lvSub3  = .LvSubItem~new(0, 3, "ts@google.com", 11)
    lvFullRow = .LvFullRow~new(lvItem, lvSub1, lvSub2, lvSub3, .true)
    list~addFullRow(lvFullRow)

    ...
```

15.125.3. imageIndex (Attribute)

```
>>--imageIndex-----------------------------------><

>>--imageIndex = varName-------------------------><
```

Reflects the index in the small icon *image* list for this subitem.

imageIndex get:

Returns this subitem's icon image index, or one of the image *constants* supplied by the **`ListView`** class, IMAGENONE or IMAGECALLBACK.

imageIndex set:

Set this attribute to the index in the small icon image list for this subitem, or to one of the image *constants* supplied by the **`ListView`** class, IMAGENONE or IMAGECALLBACK.

Remarks:

In general, if the programmer does not explicitly set this attribute, the ooDialog framework will set it to the IMAGENONE constant. When the **`LvSubItem`** object is used to retrieve information about a subitem, the list-view control does not update the value for the index if the list-view does not have the SUBITEMIMAGES *extended* style. Because of this, the value of this attribute is not well defined when the list-view does not have the SUBITEMIMAGES style.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example will set, or change, the image index for the subitem icon of the item at index 13, column 2 to 3. None of the other list-view item information will be changed.

```
lvSubItem = .LvSubItem~new(13, 2, , 3)
list~modifyItem(lvItem)
```

15.125.4. item (Attribute)

```
>>--item-----------------------------------------><

>>--item = varName-------------------------------><
```

Reflects the index of the item in the list-view that this subitem belongs to.

index get:

Returns the zero-based index of this subitem's item.

index set:

Set this attribute to a zero-based index to specify which item in the list-view this subitem belongs to.

Remarks:

If this subitem is placed into a *LvFullRow* object, the ooDialog framework will update this attribute to reflect the actual item index of the full row the subitem is placed in.

15.125.5. mask (Attribute)

```
>>--mask-----------------------------------------><

>>--mask = varName-------------------------------><
```

The *mask* attribute is used to specify which values of the **`LvSubItem`** object are valid to set or receive the information for a list-view item's subitem.

mask get:

Returns a blank separated list of keywords specifying which attributes of this subitem are valid. The possible keywords are listed in the remarks section.

mask set:

To specify which attributes in this item are to be used to set or receive information, assign a list of blank separated keywords to this attribute. The possible keywords are listed in the remarks section.

Remarks:

The *mask* attributed contains a list of blank separate keywords that specify which attributes of this **`LvSubItem`** object contain data to be set or which attributes are being requested. In general, when setting data, the ooDialog framework will try to set the *mask* attribute correctly without the programmer having to worry about this argument. As each attribute of the **`LvSubItem`** object is set, the appropriate keyword is added to the *mask* attribute.

However, if this object is going to be used to retrieve information about the underlying list-view item's subitem, the ooDialog framwork has no way to know what attributes the programmer wishes to retrieve. For this use of the **`LvSubItem`** object, the programmer should set the mask to the value that will retrieve the information he wants.

The *mask* attribute can have zero or more of the following keywords. When assigning the string to this attribute case is insignificant. When getting the value of this attribute, the returned keywords will be all upper-cased:

ALL	NORECOMPUTE
IMAGE	TEXT

ALL

A convenience keyword, the same as using the string: *IMAGE NORECOMPUTE TEXT*.

IMAGE

The *imageIndex* attribute is valid or should be set.

NORECOMPUTE

Used when the **`LvSubItem`** object will be used to retrieve information through the *getSubitem* method. Signals the list-view control should not generate a GETDISPINFO notification to retrieve the text. Rather, the *text* attribute will contain *lpStrTextCallBack*.

TEXT

The *text* attribute is valid or should be set.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example sets the icon index to 2, for the subitem in column 3 of the item at index 11. In this example, the mask attribute is set explicitly to make a point. But it is not actually needed:

```
subitem = .LvSubItem~new(11, 3, , 2, "IMAGE")

list~modifySubItem(subitem)
```

This code snippet will work exactly the same as the example above:

```
subitem = .LvSubItem~new(11, 3, , 2)

list~modifySubItem(subitem)
```

15.125.6. subItem (Attribute)

```
>>--subItem--------------------------------------><
```

```
>>--subItem = varName----------------------------><
```

Reflects the subitem index of the **`LvSubItem`** object.

subItem get:
Returns the one-based subitem index of this object.

subItem set:
Set this attribute to the one-based index of the list-view subitem this object represents.

Remarks:
If this subitem is placed into a *LvFullRow* object, the ooDialog framework will update this attribute to reflect the actual column of the full row the subitem is placed at.

15.125.7. text (Attribute)

```
>>--text-----------------------------------------><

>>--text = varName-------------------------------><
```

Reflects the text, the label, for this subitem.

text get:
Returns the text for this subitem. If no text has been set for this subitem, the **`.nil`** object is returned. Note that setting the text to the empty string is different than not having the text set at all. In addition, the operarating system uses a special value to indicate to the list-view that it should generate a GETDISPINFO notification to retrieve the text for the subitem from the application. If this value is set, the string *lpStrTextCallBack* is returned as the text for this subitem.

text set:
Assign a string to this attribute to set the text for the subitem. The string must be less than or equal to 260 characters in length. In addition the special string, *lpStrTextCallBack*, can be used to set the text value for the subitem to the value that indicates the list-view should generate a GETDISPINFO notification. Case is not significant when using *lpStrTextCallBack*.

Remarks:
Currently, there is no support for the GETDISPINFO event notification in *connectListViewEvent* method. It is anticipated that support for this event will be added in the future. Until then, using the *lpStrTextCallBack* value will simply result in the subitem appearing with no text.

Details
Raises syntax errors when incorrect usage is detected.

Example:
This example gets the current text for a subitem, appends a string to the text, and sets the text for the subitem to the new value:

```
::method addQualifier private
    expose list
    use strict arg itemIndex, subitemIndex, additionalText

    lvSubItem = .LvSubItem~new(itemIndex, subitemIndex, , , 'TEXT')
    if list~getSubItem(lvSubItem) then do
        lvSubItem~text = lvSubItem~text additionalText
        list~modifySubItem(lvSubItem)
        return .true
    end
    else do
        return .false
    end
```

Month Calendar Controls

A month calendar control provides an easy to use interface that allows users to select or enter dates. The control has the appearence of a typical calendar. The month calendar interface allows the user to select a date from the displayed days or change the control's display in various ways. The appearence of month calendar controls differ slightly between different versions of Windows.

The **`MonthCalendar`** object can not be used for dates prior to 1601 because Windows does not support dates prior to 1601. The month calendar control is based on the Gregorian calendar, which was introduced in 1753. It will not calculate dates that are consistent with the Julian calendar that was in use prior to 1753.

The **`MonthCalendar`** class provides methods that allow access to all the capabilities of the underlying Windows month calendar control, including the features available only on Windows Vista and Windows 7. To use the Windows Vista features, the Rexx program must be running on Vista or later. Likewise, to use Windows 7 features, the Rexx program must be running a Windows 7, or later, system.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with month calendar controls:

Instantiation:

Use the *newMonthCalendar* method to retrieve an object of the **`MonthCalendar`** class.

Dynamic Definition:

To dynamically create a month calendar control in the template of an *UserDialog*, use the *createMonthCalendar* method.

Event Notification

To receive notification of month calendar events use the *connectMonthCalendarEvent*) method.

One of the month calendar event notifications, the *GETDAYSTATE* notification, requires that a buffer of day states be returned in reply. The Rexx programmer can not construct the proper buffer in Rexx code. The *DayState* and *DayStates* classes are provided to help manage day states and to construct the proper buffer.

16.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with **`MonthCalendar`** objects, including other relevant classes and methods from other classes:

Table 16.1. MonthCalendar Methods and Attributes

Method	Description
Useful External Classes	
DayState	Represents the state of each day in a month.
DayStates	Manages **`DayState`** objects and constructs buffers of day states.
Useful External Methods	
new	Obtains a MonthCalendar object that represents a control in a dialog.
createMonthCalendar	Creates a MonthCalendar control in an *UserDialog*
connectMonthCalendarEvent	Connects MonthCalendar event notifications to a Rexx dialog method.

Method	Description
Attributes	
date	Reflects the currently selected date in the calendar.
Instance Methods	
addStyle	Adds the specified month calendar style(s) to the control.
getCalendarBorder	Retrieves the month calendar's border, in pixels.
getCalendarCount	Gets the number of calendars currently displayed in the month calendar control.
getCALID	Returns a keyword specifying the current calendar ID for the month calendar control.
getColor	Retrieves the color for a given portion of a month calendar control.
getCurrentView	Returns a keyword specifying the current view for the month calendar control.
getFirstDayOfWeek	Returns the first day of the week, as a whole number, for the month calendar control.
getGridInfo	Gets information about a calendar grid in the month calendar control.
getMaxSelection	Retrieves the maximum date range that can be selected in the month calendar control.
getMaxTodayWidth	Retrieves the maximum width of the *today* string in the month calendar control
getMinRect	Fills in a *Rect* object with the minimum size required to display a full month in the month calendar control.
getMonthDelta	Returns the scroll rate for the month calendar control.
getMonthRange	Retrieves date information that represents the high and low limits of a month calendar control's display.
getRange	Gets the current minimum and maximum allowable dates for the month calendar control.
getSelectionRange	Retrieves date information for the upper and lower limits of the current range selected by the user in a multi-selection month calendar control.
getStyle	Returns the current style of the specified month calendar control as list of style keywords.
getToday	Retrieves the date information for the date specified as *today* for the month calendar control.
hitTestInfo	Determines which portion of a month calendar control is at a given point on the screen.
removeStyle	Removes the specified month calendar style(s) from the control.
replaceStyle	Removes some existing month calendar style(s) and adds some new style(s).
setCalendarBorder	Sets the size, in pixels, of the month calendar control's border.
setCALID	Sets the calendar ID for the calendar control.
setColor	Sets the color for a given part of a month calendar control.
setCurrentView	Sets the view for the month calendar control.
setDayState	Sets the day states for all months that are currently visible within a month calendar control.

Method	Description
setDayStateQuick	Sets the day states for 3 months that are currently visible within a month calendar control.
setFirstDayOfWeek	Sets the first day of the week for a month calendar control.
setMaxSelection	Sets the maximum number of days that can be selected in a multi-selection month calendar control
setMonthDelta	Sets the number of months that the month calendar control moves its display when the user clicks a scroll button.
setRange	Sets the minimum and maximum allowable dates for a month calendar control.
setSelectionRange	Sets the selection for a month calendar control to a given date range.
setToday	Sets the *today* selection for a month calendar control.
sizeRectToMin	Calculates how many calendars will fit in the given rectangle and returns the minimum size a rectangle needs to be to fit the calendars.

16.2. newMonthCalendar (dialog object method)

`MonthCalendar` objects can not be instantiated by the programmer from Rexx code using a *new*() method. Rather, a month calendar object is obtained by using the *newMonthCalendar* method of the *dialog* object. The syntax is:

```
>>-newMonthCalendar(--id--+------------+--)----><
                          +-,--category-+
```

16.3. createMonthCalendar (UserDialog method)

A **`MonthCalendar`** object can be added to the dialog template in a *UserDialog* through the *createMonthCalendar* method. The basic syntax is:

```
>>-createMonthCalendar(-id-,--x-,--y-,--cx-,--cy-+----------+-+------------+--><
                                                 +-,--style-+ +-,--attrName-+
```

16.4. connectMonthCalendarEvent (dialog object method)

To connect event notifications from a month calendar control use the *connectMonthCalendarEvent* method of the *dialog* object. The basic syntax is:

```
>>-connectMonthCalendarEvent(--id--,--event--+----------------+--)------------><
                                             +--,-methodName--+
```

16.5. date (Attribute)

```
>>--date-----------------------------------------><

>>--date=varName---------------------------------><
```

The *date* attribute reflects the currently selected date in the month calendar control.

date get:

The value of the *date* attribute is a **DateTime** object that reflects the currently selected date in the control. The time portion of the **DateTime** object has no meaning. For multi-selection month calendar controls, the date is the first selected date in the range of selected dates.

date set:

A **DateTime** object must be used to set the *date* attribute. The time portion of the **DateTime** object is ignored. When the *date* attribute is assigned a new value, the currently selected date in the underlying control is updated to reflect the value of the attribute. With multi-selection month calendar controls, when the *date* attribute is assigned a new value, both the start and end of the selected range are updated to reflect the new value.

Remarks:

The *date* attribute is most useful when used with single-selection month calendar controls. For multi-selection controls, use the *getSelectionRange* and *setSelectionRange* methods to work with the full range of selected dates.

Details:

Raises syntax errors if incorrect usage is detected.

Example:

This example initializes a month calendar control with a currently selected date 10 years prior to the current day.

```
::method initDialog

  mc = self~newMonthCalendar(IDC_MC)
  mc~date = .DateTime~new~addYears(-10)
```

16.6. addStyle

```
>>--addStyle(--styles--)-------------------------><
```

Adds the specified month calendar style(s) to the control.

Arguments:

The single argument is:

style [required]

A list of 1 or more of the following month calendar style keywords, separated by spaces, case is not significant:

DAYSTATE	NOCIRCLE	SHORTDAYS
MULTI	WEEKNUMBERS	NOSELCHANGE
NOTODAY	NOTRAILING	

DAYSTATE

The month calendar will send the GETDAYSTATE *event* notification to request information about which days should be displayed in bold. Use the *connectMonthCalendarEvent* method to connect this notification with a method in the Rexx dialog object.

MULTI
: This style allows the user to select a range of dates within the control. By default, the maximum range is one week. The programmer changes the maximum range that can be selected by using the *setRange* method.

NOTODAY
: The month calendar control will not display the *today* date at the bottom of the control.

NOCIRCLE
: The month calendar control will not circle the *today* date at the bottom of the control.

WEEKNUMBERS
: The month calendar displays week numbers (1-52) to the left of each row of days. Week 1 is defined by Microsoft as the first week that contains at least four days.

NOTRAILING
: **Requires Windows Vista or later**. This style disables displaying the dates from the previous / next month in the current calendar.

SHORTDAYS
: **Requires Windows Vista or later**. With this style, the month calendar uses the shortest day name for the label of the day of the week column header.

NOSELCHANGE
: **Requires Windows Vista or later**. With this style the selection does not change when the user navigates to the next or previous month in the calendar. This style is needed for the user to be able to select a range greater than what is currently displayed.

Return value:

A list of month calendar style keywords, separated by spaces, that was the style of the month calendar control before the *replaceStyle* method was invoked. The list could be empty if the previous style was the default month calendar style.

Details:

Sets the *.systemErrorCode*.

Example:

This example adds the NOSELCHANGE style to the calendar, provide the Rexx program is running on Vista or later.

```
::method initDialog

  ...

  if .OS~isAtLeastVista then do
    mc = self~newMonthCalendar(IDC_MC)
    mc~addStyle("NOSELCHANGE")
  end
```

16.7. getCalendarBorder

```
>>--getCalendarBorder----------------------------><
```

Retrieves the month calendar's border, in pixels.

Arguments:

There are no arguments to this method.

Return value:

The return is the width of the month calendar border in pixels.

Details:

Requires Windows Vista or later.

Raises syntax errors if incorrect usage is detected.

16.8. getCalendarCount

```
>>--getCalendarCount-----------------------------><
```

Gets the number of calendars currently displayed in the month calendar control.

Arguments:

This method does not use any arguments.

Return value:

Returns the number of calendars displayed.

Details:

Requires Windows Vista or later.

Raises syntax errors if incorrect usage is detected.

16.9. getCALID

```
>>--getCalID-------------------------------------><
```

Returns a keyword specifying the current calendar ID for the month calendar control.

Arguments:

There are no arguments to this method.

Return value:

A calendar ID keyword. The possible keywords and Microsoft's explanations for the keywords are listed in the Remarks section.

Remarks:

This table lists the keywords that might be returned from this method:

Table 16.2. Calendar IDs

ID Keyword	Description
GREGORIAN	Gregorian (localized)
GREGORIAN_US	Gregorian (English strings always
JAPAN	Japanese Emperor Era

ID Keyword	Description
TAIWAN	Taiwan calendar
KOREA	Korean Tangun Era
HIJRI	Hijri (Arabic Lunar)
THAI	Thai
HEBREW	Hebrew (Lunar)
GREGORIAN_ME_FRENCH	Gregorian Middle East French
GREGORIAN_ARABIC	Gregorian Arabic
CAL_GREGORIAN_XLIT_ENGLISH	Gregorian transliterated English
CAL_GREGORIAN_XLIT_FRENCH	Gregorian transliterated French
UMALQURA	Um Al Qura (Arabic lunar) calendar

Details:

Requires Windows Vista or later.

Raises syntax errors if incorrect usage is detected.

16.10. getColor

```
>>--getColor(--calendarPart--)-------------------><
```

Retrieves the color for a given portion of a month calendar control.

Arguments:

The single argument is:

calendarPart [required]

A single keyword specifying which part of the calendar to get the color for, case is not significant. The valid keywords are:

BACKGROUND	TITLEBK
MONTHBK	TITLETEXT
TEXT	TRAILINGTEXT

BACKGROUND

Retrieves the background color displayed between months.

MONTHBK

Retrieves the background color displayed within the month.

TEXT

Retrieves the color used to display text within a month.

TITLEBK

Retrieves the background color displayed in the calendar's title.

TITLETEXT

Retrieves the color used to display text within the calendar's title.

TRAILINGTEXT

Retrieves the color used to display header day and trailing day text. Header and trailing days are the days from the previous and following months that appear on the current month calendar.

Return value:

On success, returns a COLORREF representing the color setting for the part of the month calendar control specified. returns CLR_INVALID on error.

Remarks:

The *Image* class has a number of convenience methods for working with COLORREFs, including the *colorRef* method that explains what a COLORREF is.

The *colorRef* method can be used to test the return for error. I.e.: **`.Image~colorRef(CLR_INVALID)`**. (An error is not very likely.)

Note that many of the underlying dialog controls ignore their color settings when visual themes are enabled, unless the theme is Windows Classic. This is the case for the month calendar control.

Details:

Raises syntax errors if incorrect usage is detected.

Example:

This example gets the color for the head and trailing day text, and checks for an error.

```
monthCalendar = self~newMonthCalendar(IDC_MC)
invalid = .Image~colorRef(CLR_INVALID)

color = monthCalendar~getColor("TRAILINGTEXT")
if color == invalid then do
  -- Fill in with some error routine.
end
```

16.11. getCurrentView

```
>>--getCurrentView-------------------------------><
```

Returns a keyword specifying the current view for the month calendar control.

Arguments:

This method does not have any arguments.

Return value:

The return will be exactly one of the following keywords describing the current view: *Monthly*, *Annual*, *Decade*, or *Century*.

Details:

Requires Windows Vista or later.

Raises syntax errors if incorrect usage is detected.

16.12. getFirstDayOfWeek

```
>>--getFirstDayOfWeek(--+--------+--)------------><
                        +--info--+
```

Returns the first day of the week, as a whole number, for the month calendar control.

Arguments:

The single argument is:

info [optional]

A **`Directory`** object in which additional information is returned.

Return value:

Returns a whole number that represents the first day of the week, where 0 is Monday, 1 is Tuesday, and so on.

If the optional *info* argument is supplied, the **`Directory`** object will contain the following indexes on return:

DAY

The numerical first day of the week. This is the same value as the return value of the method.

USINGLOCALE

True if the calendar is using the localized first day of the week, false if the first day of the week has been set to something other than the localized first day of the week.

DAYNAME

The english name of the day corresponding to first day of the week.

Details:

Raises syntax errors when incorrect arguments are detected.

16.13. getGridInfo

```
>>--getGridInfo(--info--)------------------------><
```

Gets information about a calendar grid in the month calendar control. A **`Directory`** object with indexes specifying the information requested is passed into the method. On return, the **`Directory`** object contains indexes with the information requested.

Arguments:

The single argument is:

info [required] [in / out]

A **`Directory`** object. On input, the object contains the following indexes specifying the information needed:

PART [required]

A single keyword, specifying which part of the calendar the information is requested for. The following keywords are accepted, case is not significant:

CONTROL	FOOTER	BODY
NEXT	CALENDAR	ROW
PREV	HEADER	CELL

CONTROL

The entire month calendar control, which may include up to 12 calendars.

NEXT
: The next button.

PREV
: The previous button.

FOOTER
: The footer.

CALENDAR
: One specific calendar. The value of the INDEX index specifies which calendar.

HEADER
: The calendar header. The value of the INDEX index specifies which calendar.

BODY
: The calendar body. The value of the INDEX index specifies which calendar.

ROW
: A given calendar row. The value of the INDEX index specifies which calendar.

CELL
: A given calendar cell. The value of the INDEX index specifies which calendar.

WHAT [required]
: A list of one or more of the following keywords, specifying what information (what indexes) will be filled in for the calendar part specified in the PART index. Case is not significant:

DATE	RECT	NAME

DATE
: The STARTDATE and ENDDATE indexes will be present and contain valid values.

RECT
: The RECT index will be present.

NAME
: The NAME index will be present.

INDEX [optional]
: Specifies the one-based index of the calendar the requested information is for. Used when the PART index specifies CALENDAR, HEADER, BODY, ROW, or CELL. If omitted, the default is 1.

ROW [optional]
: If the PART keyword is ROW, specifies the row for which to return information.

COL [optional]
: If the PART keyword is CELL, specifies the column of the cell for which to return information. The ROW index provides the row of the cell for which to return information.

On a successful return the **`Directory`** object contains the following indexes, dependent on the information requested:

SELECTED

If PART is CELL, indicates if the cell described by the values in the ROW and COL is currently selected. The value of the SELECTED index will be **`.true`** if the cell is selected and **`.false`** if it is not.

STARTDATE

Contains a **`DateTime`** object representing the start date of the Calendar specified by INDEX index. Only valid when the WHAT index contains the DATE keyword.

ENDDATE

Contains a **`DateTime`** object representing the end date of the Calendar specified by INDEX index. Only set when the WHAT index contains the DATE keyword.

RECT

A *Rect* object containing the dimensions of the calendar part specified in the PART index. Only set when the WHAT index contains the RECT keyword.

NAME

Set only if the WHAT index contains the NAME keyword, and only for the following PART keywords:

CALENDAR:

Returns the text of the selected dates. In the case of multiple selection, returns the text for the date at the start of the selection.

CELL:

Returns the text of the cell indicated by ROW and COL indexes, for instance *15*, if the 15th day was specified.

HEADER:

Returns the text in the calendar header, for instance *March 1951*.

Return value:

True on success, otherwise false.

Details:

Requires Windows Vista or later.

Raises syntax errors when incorrect usage is detected.

Example:

This contrived example determines the date of the cell at row 2, column 3 in the second displayed calendar, if there is a second calendar. It also determines if the cell is selected and its dimensions:

```
if .OS~isAtLeastVista then do
  mc = self~newMonthCalendar
  if mc~getCalendarCount >= 2 then do
    info = .directory~new
    info~part = "CELL"
    info~what = "RECT NAME"
    info~index = 2
    info~row   = 2
    info~col   = 3

    if mc~getGridInfo(info) then do
      selectedWords = "is not selected,"
      if info~selected then selectedWords = "is selected."
```

```
        width  = info~rect~right - info~rect~left
        height = info~rect~bottom - info~rect~top

        say "The date of the day at row 2, column 3 in the second calendar is"
 info~name"."
        say "The day" selectedWords
        say "The day is at ("info~rect~left"," info~rect~top")" on the screen."
        say "It is" width "pixels wide and" height "pixels high."
      end
      else do
        say "Unexpected failure of getGridInfo()."
      end
    end
  end
```

16.14. getMaxSelection

```
>>--getMaxSelection------------------------------><
```

Retrieves the maximum date range that can be selected in the month calendar control.

Arguments:

There are no arguments for this method.

Return value:

Returns the total number of days that can be selected for the month calendar control.

Remarks:

The maximum date range that can be selected can be changed by using the *setMaxSelection* method.

16.15. getMaxTodayWidth

```
>>--getMaxTodayWidth-----------------------------><
```

Retrieves the maximum width of the *today* string in the month calendar control, which includes the date and label text.

Arguments:

This method does not have any arguments.

Return value:

Returns the width of the today string in pixels

16.16. getMinRect

```
>>--getMinRect(--rect--)-------------------------><
```

Fills in a *Rect* object with the minimum size required to display a full month in the month calendar control.

Arguments:

The single argument is:

rect [required]

A **`Rect`** object in which to return the minimum size.

Return value:

If the method succeeds, **`.true`** is returned, otherwise **`.false`**.

Remarks:

The minimum required size for the month calendar control depends on things like the current font, the style in effect for the month calendar, etc.. If the application changes anything that will affect the minimum size, the programmer should use the *getMinRect* method to recalculate the minimum size.

Note that the rectangle returned will not include the width of the *Today* string. If the month calendar does not have the NOTODAY style set, the programmer should also get the *Today* string width by using the *getMaxTodayWidth* method. For the width portion of the minimum required size, use the larger of the value reported by *getMaxTodayWidth* and the value reported by *getMinRect*. This will ensure that the *Today* string is not clipped..

The *top* and *left* of the **`Rect`** object will always be 0. The *right* and *bottom* of the **`Rect`** object will contain the minimum width and height required for the month calendar.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

This example resizes a month calendar control to the minimum required size. The position of the control, i.e., the upper left hand corner, remains unchanged.

```
::method sizeCalendar private
  use strict arg calendar

  r = .Rect~new
  if calendar~getMinRect(r) then calendar~resizeTo(.Size~new(r~right, r~bottom))
```

16.17. getMonthDelta

```
>>--getMonthDelta--------------------------------><
```

Returns the scroll rate for the month calendar control. The scroll rate is the number of months that the control moves its display when the user clicks a scroll button.

Arguments:

This method does not use any arguments.

Return value:

When the scroll rate has been previously set through the *setMonthDelta* method, the return is a whole number specifying the current scroll rate. If, the scroll rate has not been previously set, of it it has been reset to the default, the return is a whole number specifying the number of months currently being displayed by the month calendar control.

16.18. getMonthRange

```
>>--getMonthRange(--range--+---------+--)--------><
                           +-,-span--+
```

Retrieves date information (using **`DateTime`** objects) that represents the high and low limits of a month calendar control's display.

Arguments:

The arguments are:

range [required] [IN/OUT]

An **`Array`** object in which the range is returned. The lower limit will be returned at index 1 and the upper limit will be returned at index 2. Both the values at the indexes will be **`DateTime`** objects.

range [optional]

A keyword specifying whether the range should include only months that are ENTIRELY displayed or to include trailing and following months that are only PARTIALLY displayed. The default if omitted is PARTIALLY. Only the first letter of ENTIRELY or PARTIALLY are required and case is insignificant.

Return value:

The return is a whole number representing the span of months of the lower and upper limits returned in the *range* argument.

Details:

Raises syntax errors when incorrect usage is detected.

16.19. getRange

```
>>--getRange(--range--)--------------------------><
```

Gets the current minimum and maximum allowable dates for the month calendar control.

Arguments:

The single argument is:

range [required] [IN/OUT]

An **`Array`** object in which the minimum and maximum dates are returned as **`DateTime`** objects. The minimum date will be at index 1 and the maximum at index 2. If the value at either index is set to zero, then no corresponding limit is set for the month calendar control.

Return value:

A keyword indicating which, if any, of the limits have been set. The keyword will be exactly one of the following:

none

The month calendar has neither a minimum nor maximum date set.

min

The month calendar has a minimum allowable date set, but no maximum date is set.

max

The month calendar has a maximum allowable date set, but no minimum date is set.

none

The month calendar has both a minimum and a maximum allowable date set.

error

An error occurred, the values at index 1 and 2 of the *range* argument are undefined. It is unlikely that an error will occur.

Remarks:

Additional comments.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

This example checks for minimum and maximum allowable dates and displays the results:

```
mc = self~newMonthCalendar(IDC_MC)
range = .array~new(2)

result = mc~getRange(range)
select
  when result == 'none' then say "No minimum or maximum allowable dates set."

  when result == 'min' then do
    say "There is no maximum allowable date set."
    say "The minimum allowable date is" self~formatDate(range[1])"."
  end

  when result == 'max' then do
    say "There is no minimum allowable date set."
    say "The maximum allowable date is" self~formatDate(range[2])"."
  end

  when result == 'both' then do
    say "The minimum allowable date is" self~formatDate(range[1])"."
    say "The maximum allowable date is" self~formatDate(range[2])"."
  end

  when result == 'error' then do
    say "An error occurred."
    say "Hang the documentation author!!!"
  end
end
```

16.20. getSelectionRange

```
>>--getSelectionRange(--range--)------------------><
```

Retrieves date information that represents the upper and lower limits of the date range currently selected by the user in a multi-selection month calendar control.

Arguments:

The single argument is:

range [required] [IN/OUT]

An **`Array`** object in which the lower and upper limits of the selection are returned as **`DateTime`** objects. The lower bound of the selection will be at index 1 and the upper bound at index 2.

Return value:

Returns **`.true`** on success, otherwise **`.false`**.

On success index 1 of the *range* array argument will contain a **`DateTime`** object whose date portion is the lower limit of the selection range. Likewise, index 2 will contain a **`DateTime`** object whose date portion is the upper limit of the selection. On error, the *range* argument is unchanged.

Remarks:

The *getSelectionRange* method is for month calendar controls that have the MULTI (multi-selection) style. If the month calendar is a single-selection calendar this method returns **`.false`**. For a single-selection month calendar the *date* attribute will reflect the current selection.

Details:

Raises syntax errors when incorrect arguments are detected.

16.21. getStyle

```
>>--getStyle-------------------------------------><
```

Returns the current style of the specified month calendar control as list of style keywords.

Arguments:

This method has no arguments.

Return value:

A list of month calendar style keywords, separated by spaces. The list could be empty if the previous style was the default month calendar style. Otherwise, the list will contain one or more of the following keywords:

DAYSTATE	NOCIRCLE	SHORTDAYS
MULTI	WEEKNUMBERS	NOSELCHANGE
NOTODAY	NOTRAILING	

DAYSTATE

The month calendar will send the GETDAYSTATE *event* notification to request information about which days should be displayed in bold. Use the *connectMonthCalendarEvent* method to connect this notification with a method in the Rexx dialog object.

MULTI

This style allows the user to select a range of dates within the control. By default, the maximum range is one week. The programmer changes the maximum range that can be selected by using the *setRange* method.

NOTODAY

The month calendar control will not display the *today* date at the bottom of the control.

NOCIRCLE

The month calendar control will not circle the *today* date at the bottom of the control.

WEEKNUMBERS

The month calendar displays week numbers (1-52) to the left of each row of days. Week 1 is defined by Microsoft as the first week that contains at least four days.

NOTRAILING
: **Requires Windows Vista or later**. This style disables displaying the dates from the previous / next month in the current calendar.

SHORTDAYS
: **Requires Windows Vista or later**. With this style, the month calendar uses the shortest day name for the label of the day of the week column header.

NOSELCHANGE
: **Requires Windows Vista or later**. With this style the selection does not change when the user navigates to the next or previous month in the calendar. This style is needed for the user to be able to select a range greater than what is currently displayed.

Example:

This example displays the current style of a month calendar control:

```
::method printStyle private

  mc = self~newMonthCalendar(IDC_MC_HOLIDAYS
  say 'The holidays month calendar has this style:' mc~getStyle
```

16.22. getToday

```
>>--getToday-------------------------------------><
```

Retrieves the date information for the date specified as *today* for a month calendar control.

Arguments:

This method does not have any arguments.

Return value:

On success, the return is a **`DateTime`** object whose date portion is the date of the *Today* date for the month calendar. On error the **`.nil`** object is returned.

16.23. hitTestInfo

```
>>--hitTestInfo(--pt--)--------------------------><
```

Determines which portion of a month calendar control is at the given point. The point is specifed in client *coordinates*.

Arguments:

The single argument is:

pt [required]
: A *Point* object containing the point to test. The point is specifed in client *coordinates*.

Return value:

The return is a **`directory`** object containing information on the point hit-tested. The **`directory`** will contain some or all of the following indexes, depending on the results of the hit test:

HIT:

This index will always be present. Its value will be exactly one of the following keywords:

CalendarBackground:

The point, *pt*, was in the calendar's background.

CalendarControl:

The point, *pt*, is outside of any calendar but within the calendar controls rectangle.

CalendarDate:

The point, *pt*, was on a particular date within the calendar. The DATE index of the **`directory`** object is set to the date at the given point.

CalendarDateMin:

The point, *pt*, was over the minimum date(s) in the calendar.

CalendarDateMax:

The point, *pt*, was over the maximum date(s) in the calendar.

CalendarDateNext:

The point, *pt*, was over a date from the next month (partially displayed at the end of the currently displayed month). If the user clicks here, the month calendar will scroll its display to the next month or set of months.

CalendarDatePrev:

The point, *pt*, was over a date from the previous month (partially displayed at the end of the currently displayed month). If the user clicks here, the month calendar will scroll its display to the previous month or set of months.

CalendarDay:

The point, *pt*, was over a day abbreviation, like *Fri*, for example. The DATE index of the **`directory`** object is set to the corresponding date in the top row.

CalendarWeekNum:

The point, *pt*, was over a week number. This only applies when the month calendar has the WEEKNUMBERS style. The DATE index of the **`directory`** object is set to the corresponding date in the leftmost column.

NoWhere:

The point, *pt*, was not on the month calendar control, or it was in an inactive portion of the control.

TitleBackground:

The point, *pt*, was over the background of a month's title.

TitleButtonNext:

The point, *pt*, was over the button at the top right corner of the control. If the user clicks here, the month calendar will scroll its display to the next month or set of months.

TitleButtonPrev:

The point, *pt*, was over the button at the top left corner of the control. If the user clicks here, the month calendar will scroll its display to the previous month or set of months.

TitleMonth:

The point, *pt*, was in a month's title bar, over a month name.

TitleYear:
The point, *pt*, was in a month's title bar, over the year value.

TodayLink:
The point, *pt*, was on the *today* link at the bottom of the month calendar control.

DATE:
When set, the DATE index will contain a **`DateTime`** object that reflects the date as specified by the HIT index above. That is, for CalendarDate, CalendarDay, and CalendarWeekNum. The index is not set for any other hit.

RECT:
Vista or later only: The index will be a *Rect* object that specifies the rectangle of the hit-tested location.

OFFSET:
Vista or later only: The OFFSET index is the one-based offset of the calendar at the hit-teste point. This is primarily for use when more than one calendar is displayed.

ROW:
Vista or later only: The one-based index of the row in the calendar grid that *pt* is over. When the point is over the calendar *grid*, which is the grid of dates, both the ROW and COLUMN indexes are set. In addition, if the hit test is *CalendarWeekNum* the ROW index is set, but the COLUMN index is not set.

COLUMN:
Vista or later only: The one-based index of the column in the calendar grid that *pt* is over. When the point is over the calendar *grid*, which is the grid of dates, both the ROW and COLUMN indexes are set. In addition, if the hit test is *CalendarDay* the COLUMN index is set, but the ROW index is not set.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

This example gets the current cursor position, hit tests that point, and then prints out the hit test information:

```
::method onF7
  expose calendar

  pt = self~getCursorPos
  calendar~screen2client(pt)

  d = calendar~hitTestInfo(pt)
  say 'Hit:    ' d~hit
  say 'Date:   ' d~date
  if d~rect == .nil then say 'Rect:   ' d~rect
  else say 'Rect:   ' '('d~rect~left',' d~rect~top',' d~rect~right',' d~rect~bottom')'
  say 'Offset:' d~offset
  say 'Row:    ' d~row
  say 'Column:' d~column
  say

/*  Output for 2 hit tests might for instance be:

Hit:    CalendarDate
Date:   2011-10-12T00:00:00.000000
Rect:   (116, 79, 147, 94)
```

```
Offset: 1
Row:    3
Column: 4

Hit:    TodayLink
Date:   The NIL object
Rect:   (63, 139, 184, 156)
Offset: 1
Row:    The NIL object
Column: The NIL object

 */
```

16.24. removeStyle

```
>>--removeStyle(--styles-)-----------------------><
```

Removes the specified month calendar style(s) from the control.

Arguments:

The single argument is:

style [required]

A list of 1 or more of the following month calendar style keywords, separated by spaces, case is not significant:

DAYSTATE	NOCIRCLE	SHORTDAYS
MULTI	WEEKNUMBERS	NOSELCHANGE
NOTODAY	NOTRAILING	

DAYSTATE

The month calendar will send the GETDAYSTATE *event* notification to request information about which days should be displayed in bold. Use the *connectMonthCalendarEvent* method to connect this notification with a method in the Rexx dialog object.

MULTI

This style allows the user to select a range of dates within the control. By default, the maximum range is one week. The programmer changes the maximum range that can be selected by using the *setRange* method.

NOTODAY

The month calendar control will not display the *today* date at the bottom of the control.

NOCIRCLE

The month calendar control will not circle the *today* date at the bottom of the control.

WEEKNUMBERS

The month calendar displays week numbers (1-52) to the left of each row of days. Week 1 is defined by Microsoft as the first week that contains at least four days.

NOTRAILING

Requires Windows Vista or later. This style disables displaying the dates from the previous / next month in the current calendar.

SHORTDAYS

Requires Windows Vista or later. With this style, the month calendar uses the shortest day name for the label of the day of the week column header.

NOSELCHANGE

Requires Windows Vista or later. With this style the selection does not change when the user navigates to the next or previous month in the calendar. This style is needed for the user to be able to select a range greater than what is currently displayed.

Return value:

A list of month calendar style keywords, separated by spaces, that was the style of the month calendar control before the *removeStyle* method was invoked. The list could be empty if the previous style was the default month calendar style.

Details:

Sets the *.systemErrorCode*.

16.25. replaceStyle

```
>>--replaceStyle(--oldStyles--,--newStyles--)----><
```

The *replaceStyle* method removes some existing month calendar styles and adds some new styles.

Arguments:

The arguments are:

oldStyles [required]

A list of 1 or more of the following month calendar style keywords, separated by spaces, case is not significant. The styles specified will be removed from the month calendar.

DAYSTATE	NOCIRCLE	SHORTDAYS
MULTI	WEEKNUMBERS	NOSELCHANGE
NOTODAY	NOTRAILING	

DAYSTATE

The month calendar will send the GETDAYSTATE *event* notification to request information about which days should be displayed in bold. Use the *connectMonthCalendarEvent* method to connect this notification with a method in the Rexx dialog object.

MULTI

This style allows the user to select a range of dates within the control. By default, the maximum range is one week. The programmer changes the maximum range that can be selected by using the *setRange* method.

NOTODAY

The month calendar control will not display the *today* date at the bottom of the control.

NOCIRCLE

The month calendar control will not circle the *today* date at the bottom of the control.

WEEKNUMBERS

The month calendar displays week numbers (1-52) to the left of each row of days. Week 1 is defined by Microsoft as the first week that contains at least four days.

NOTRAILING

Requires Windows Vista or later. This style disables displaying the dates from the previous / next month in the current calendar.

SHORTDAYS

Requires Windows Vista or later. With this style, the month calendar uses the shortest day name for the label of the day of the week column header.

NOSELCHANGE

Requires Windows Vista or later. With this style the selection does not change when the user navigates to the next or previous month in the calendar. This style is needed for the user to be able to select a range greater than what is currently displayed.

newStyles [required]

A list of 1 or more month calendar style keywords, separated by spaces, case is not significant. The styles specified will be added to the month calendar. The acceptable keywords are the same as those for the *oldStyles* argument above.

Return value:

A list of month calendar style keywords, separated by spaces, that was the style of the month calendar control before the *replaceStyle* method was invoked. The list could be empty if the previous style was the default month calendar style.

Remarks:

The *replaceStyle* method is essentially a combination of the *removeStyle* and *addStyle* methods into one convenience method.

Details:

Sets the *.systemErrorCode*.

Example:

This example checks if the Rexx program is running on a Vista or later system. If it is, it removes the NOTODAY and NOCIRCLE styles and adds the Vista specific, NOSELCHANGE, SHORTDAYS, and NOTRAILING styles.

```
::method initDialog

  ...

  if .OS~isAtLeastVista then do
    mc = self~newMonthCalendar(IDC_MC)
    mc~replaceStyle("NOTODAY NOCIRCLE", "NOSELCHANGE SHORTDAYS NOTRAILING")
  end
```

16.26. setCalendarBorder

```
>>--setCalendarBorder(--+----------+--)---------><
                        +--pixels--+
```

Sets the size, in pixels, of the month calendar's border.

Arguments:

The single argument is:

pixels [optional]

The size to set the border to, in pixels. If *pixels* is omitted, the border is set to the default value specified by the theme, or zero if themes are not being used.

Return value:

Returns 0, always.

Details:

Requires Windows Vista or later.

Raises syntax errors when incorrect arguments are detected.

Example:

This example sets the calendar border to a very large size and then resizes the calendar to its optimal size:

```
::method sizeCalendar private
  use strict arg calendar

  calendar~setCalendarBorder(32)

  r = .Rect~new
  if \ calendar~getMinRect(r) then return 0

  calendar~resizeTo(.Size~new(r~right, r~bottom))
  ...
```

16.27. setCALID

```
>>--setCALID(--calID--)--------------------------><
```

Sets the calendar ID for the calendar.

Arguments:

The single argument is:

calID [required]

A single keyword specifying the calendar ID to use. The keywords are case insensitive. The keywords are listed in the following table:

Table 16.3. Calendar ID Keywords

ID Keyword	Description
GREGORIAN	Gregorian (localized)
GREGORIAN_US	Gregorian (English strings always
JAPAN	Japanese Emperor Era
TAIWAN	Taiwan calendar
KOREA	Korean Tangun Era
HIJRI	Hijri (Arabic Lunar)
THAI	Thai
HEBREW	Hebrew (Lunar)
GREGORIAN_ME_FRENCH	Gregorian Middle East French

ID Keyword	Description
GREGORIAN_ARABIC	Gregorian Arabic
CAL_GREGORIAN_XLIT_ENGLISH	Gregorian transliterated English
CAL_GREGORIAN_XLIT_FRENCH	Gregorian transliterated French
UMALQURA	Um Al Qura (Arabic lunar) calendar

Return value:

Returns 0, always.

Details:

Requires Windows Vista or later.

Raises syntax errors when incorrect arguments are detected.

16.28. setColor

```
>>--setColor(--which--,--color--)----------------><
```

Sets the color for a given part of a month calendar control.

Arguments:

The arguments are:

which [required]

Specifies which part of the calendar will have its color set. Exactly one of the following keywords must be used, case is not significant:

BACKGROUND	TEXT	TITLETEXT
MONTHBK	TITLEBK	TRAILINGTEXT

BACKGROUND

Set the background color displayed between months.

MONTHBK

Set the background color displayed within the month.

TEXT

Set the color used to display text within a month.

TITLEBK

Set the background color displayed in the calendar's title.

TITLETEXT

Set the color used to display text within the calendar's title.

TRAILINGTEXT

Set the color used to display header day and trailing day text. Header and trailing days are the days from the previous and following months that appear on the current month calendar.

color [required]

The color, specified as a COLORREF, to set the specified calendar part.

Return value:

On success, returns the previouse color, as a COLORREF, for the part of the month calendar control specified. Returns CLR_INVALID on error.

Remarks:

The *Image* class has a number of convenience methods for working with COLORREFs, including the *colorRef* method that explains what a COLORREF is.

The *colorRef* method can be used to test the return for error. I.e.: **`.Image~colorRef(CLR_INVALID)`**. (An error is not very likely.)

Note that many of the underlying dialog controls ignore their color settings when visual themes are enabled, unless the theme is Windows Classic. This is the case for the month calendar control. Setting any of the calendar part colors will have no effect, except in the Windows Classic theme.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

This example sets the color for the TEXT calendar part to blue and prints out the RGB values for the previous color.

```
monthCalendar = self~newMonthCalendar(IDC_MC_HOLIDAYS)
invalid = .Image~colorRef(CLR_INVALID)

-- Blue
color = .Image~colorRef(255, 0, 0)

oldColor = monthCalendar~setColor("TEXT", color)
if oldColor == invalid then do
  -- Fill in with some error routine.
  say 'setColor() failed.'
end
else do
  R = .Image~getRValue(oldColor)
  G = .Image~getGValue(oldColor)
  B = .Image~getBValue(oldColor)
  say 'The old color for "TEXT" was:('R',' G',' B')'
end
```

16.29. setCurrentView

```
>>--setCurrentView(--view--)---------------------><
```

Sets the view for a month calendar control.

Arguments:

The single required argument is:

view [required]

The *view* argument must be exactly one of the following keywords, case is not significant:

MONTHLY	ANNUAL	DECADE	CENTURY

MONTHLY

Monthly view.

ANNUAL
: Annual view.

DECADE
: Decade view.

CENTURY
: Century view.

Return value:

Returns true on success, false on error.

Details:

Raises syntax errors when incorrect arguments are detected.

Requires Windows Vista or later.

16.30. setDayState

```
>>--setDayState(--dayStates--)-------------------><
```

Sets the day states for all months that are currently visible within a month calendar control.

Arguments:

The single argument is:

dayStates [required]
: An array of *DayState* objects. The array must not be sparse and can not contain more items than the number of currently visible months in the calendar. Index 1 in the array should contain the **`DayState`** object for the first visible month in the calendar. Each consecutive index contains the **`DayState`** object for the next visible month, with the last index in the array containing the **`DayState`** object for the last visible month in the calendar.

Return value:

Returns true on success, false on error.

Remarks:

A month calendar uses a day state to determine how to display individual dates in a month. The day state specifies which days should be bolded. The month calendar must have the *createMonthCalendar* style or setting the day state has no effect. The programmer must use the *DayState* class to create the day state objects.

The number of elements in *dayStates* must equal the return value from *getMonthRange* when used with the PARTIALLY keyword argument. That is, *dayStates* must contain values that correspond with all months currently in the control's display, in chronological order. This includes the two months that may be partially displayed before the first month and after the last month.

Normally in a program using day states with a month calendar, the programmer will not have to continually invoke the *setDayState* method as the user scrolls through the calendar. Rather he will connect the *connectMonthCalendar* event of the month calendar and return an array of day state objects from the *GETDAYSTATE* event handler. The month calendar then sets the day states internally using the array.

However, the programmer will need to use the *setDayState* method to properly display the dates when the dialog with a month calendar is first shown.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

This example sets the 15th and 30th of each month to be bolded in the month calendar when the dialog is first shown. Notice that the day state for each month is the same (15th and 30th) so the code does not determine exactly which months are showing, it only needs to determine how many months are showing.

```
::method initDialog
  expose calendar

  calendar = self~newMonthCalendar(IDC_MC_PAYDAYS)

  monthsShowing = .array~new
  count = calendar~getMonthRange(monthsShowing, 'P')

  dayStates = .array~new(count)
  dayState = .DayState~new(15, 30)
  do i = 1 to count
    dayStates[i] = dayState
  end

  calendar~setDayState(dayStates)
  ...
```

16.31. setDayStateQuick

```
>>--setDayStateQuick(--ds1-,-ds2-,-ds3--)--------><
```

Sets the day states for 3 months that are currently visible (assumed) within a month calendar control. The *setDayStateQuick* method is almost identical to the *setDayState* method except that the number of *DayState* state objects is fixed at 3.

Arguments:

The arguments are:

ds1
: The *DayState* object for the first month.

ds2
: The *DayState* object for the second month.

ds3
: The *DayState* object for the third month.

Return value:

xx

Remarks:

A month calendar uses a day state to determine how to display individual dates in a month. The day state specifies which days should be bolded. The month calendar must have the *createMonthCalendar* style or setting the day state has no effect. The programmer must use the *DayState* class to create the day state objects.

The most common use case for a month calendar is to display 1 month. In this case, when setting the day states for the calendar, exactly 3 months of day state objects are needed, (previouse, current, and next months.) The *setDayStateQuick* is simply a short cut method to take advantage of this foreknowledge.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

This example sets the day states for the month calendar when the dialog is initially displayed. Since the programmer knows the calendar is set to only display 1 month, she knows that she need exactly 3 day state objects. 1 for the displayed month, and 1 each for the leading and trailing days of the previous and next months:

```
::method initDialog
  expose calendar

  calendar = self~newMonthCalendar(IDC_MC_PAYDAYS)

  dayState = .DayState~new(15, 30)
  calendar~setDayStateQuick(dayState, dayState, dayState)
  ...
```

16.32. setFirstDayOfWeek

```
>>--setFirstDayOfWeek(--firstDay--)--------------><
```

Sets the first day of the week for the month calendar control.

Arguments:

The single required argument is:

firstDay [required]

Which day is to be the first day of the week. This can either be the name of the day (Monday, Tuesday, etc., case insignificant) or the number of the day (0 for Monday, 1 for Tuesday, etc..)

Return value:

A **`Directory`** object with information concerning the previous first day of the week. This is the same information as returned by the *getFirstDayOfWeek* method.

Remarks:

If the first day of the week is set to anything other than the default, which is the locale dependent first day of the week, the control will no longer automatically update first-day-of-the-week changes based on locale changes.

The returned **`Directory`** object will contain the following indexes with information concerning ther previous first day of the week:

DAY

The numerical first day of the week (0 for Monday, 1 for Tuesday, etc..)

USINGLOCALE

True if the calendar is using the localized first day of the week, false if the first day of the week has been set to something other than the localized first day of the week.

DAYNAME
The english name of the day corresponding to first day of the week.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

16.33. setMaxSelection

```
>>--setMaxSelection(--count--)-------------------><
```

Sets the maximum number of days that can be selected in a month calendar control.

Arguments:

The single required argument is:

count [required]
The maximum number of days that can be selected.

Return value:

Returns true on success, otherwise false.

Remarks:

This method will fail if the month calendar control does not have the multi-select style.

Details:

Raises syntax errors when incorrect arguments are detected.

16.34. setMonthDelta

```
>>--setMonthDelta(--amount--)--------------------><
```

Sets the number of months that the month calendar control moves its display when the user clicks a scroll button. This is called the scroll rate.

Arguments:

The single required argument is:

amount [required]
The number of months to be set as the month calendar's scroll rate. If *amount* is zero, the month delta is reset to the default. The defualt is the number of months displayed in the month calendar control.

Return value:

Returns the previous scroll rate. If the scroll rate was not previously set, the return value is zero

Remarks:

The PAGE UP and PAGE DOWN keys, (**.VK~PRIOR** and **.VK~NEXT** of the *VK* class,) change the selected month by one, regardless of the number of months displayed or the value set by *setMonthDelta*.

Details:

Raises syntax errors when incorrect arguments are detected.

16.35. setRange

```
>>--setRange(--dates--)--------------------------><
```

Sets the minimum and maximum allowable dates for the month calendar control.

Arguments:

The single required argument is:

dates [required]

An array of **`DateTime`** objects used to set the minimum and maximum dates. The **`DateTime`** object at index 1 sets the minimum date and the **`DateTime`** object at index 2 sets the maximum date.

Return value:

xx

Remarks:

The array must contain at least one of the indexes. If it contains neither, and exceptions is raised. If one of the array indexes is empty, then the corresponding date is not set. The time portion of the **`DateTime`** object(s) is ignored.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

This example sets a month calendar to only show the months of 2011:

```
::method initDialog
  expose calendar

  calendar = self~newMonthCalendar(IDC_MC_HOLIDAYS)
  ...

  -- Restrict the calendar so that it only displays the year 2011.
  start = .DateTime~fromStandardDate("20110101")
  end = .DateTime~fromStandardDate("20111231")

  calendar~setRange(.array~of(start, end))
```

16.36. setSelectionRange

```
>>--setSelectionRange(--dates--)-----------------><
```

Sets the selection for a multi-selection month calendar control to a given date range.

Arguments:

The single required argument is:

dates [required]

An **`Array`** object containing two **`DateTime`** objects. The **`DateTime`** object at index 1 must be the starting date of the selection and the **`DateTime`** object at index 2 must be the ending date of the selection. The time portion of the **`DateTime`** object is ignored.

Return value:

Returns true on success, false on error.

Remarks:

This method will fail if the month calendar does not have the multi-selection style.

Details:

Raises syntax errors when incorrect arguments are detected.

16.37. setToday

```
>>--setToday(--+--------+--)---------------------><
               +--date--+
```

Sets the *today* selection for a month calendar control.

Arguments:

The single optional argument is:

date [optional]

A **`DateTime`** object that contains the date to set the *today* selection to. The time portion of the object is ignored. If this argument is omitted, the month calendar reverts to the default setting for the *today* selection.

Return value:

Returns 0, always.

Remarks:

When *today* is set to anything other than the default, then these conditions apply:

- The month calendar will not automatically update the *today* selection when time passes midnight of the current day.
- The month calendar will not automatically update its display based on locale changes.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

The following example sets the *today* selection to tomorrow.

```
calendar = self~newMonthCalendar(IDC_MC_PAYDAYS)

-- Pretend it is the day after tomorrow:
date = .DateTime~today
calendar~setToday(date~addDays(2))
...
```

16.38. sizeRectToMin

```
>>--sizeRectToMin(--rect--)----------------------><
```

Calculates how many calendars will fit in the given rectangle, and then returns the minimum size that a rectangle needs to be to fit that number of calendars.

Arguments:

The single required argument is:

rect [required] [in / out]

A *Rect* object that, on entry, describes a region that is greater than or equal to the size necessary to fit the desired number of calendars. When this method returns, the **`rect`** object will contain the minimum size needed for this number of calendars.

Return value:

Returns 0, always.

Details:

Raises syntax errors when incorrect arguments are detected.

Requires Windows Vista or later.

16.39. DayState Class

A **`DayState`** object represents the state of each day in a month. It is a specialty class used by the *MonthCalendar* class when the month calendar needs to reply to the *GETDAYSTATE* event.

Each **`DayState`** object contains a single value. The value encodes the state of each day in a format understood by the *underlying* Windows month calendar control. The state of a single day is a binary value. I.e., on or off, true or false, special or not special, however it is convenient for the programmer to think of. When the state of a day is on, the month calendar displays that day in bold.

16.39.1. Method Table

The following table lists the class and instance methods of the **`DayState`** class:

Table 16.4. DayState Class Method Reference

Method	Description
Class Methods	**Class Methods**
new	Instantiates a new **`DayState`** object and sets the state of each day.
Instance Methods	**Instance Methods**
value	Returns the *value* of the **`DayState`** object.

16.39.2. new (Class Method)

```
          +--,-----+
          V        |
>>--new(---dayNum--+--)--------------------------><
```

Instantiates a new **`DayState`** object and sets the state of each day.

Arguments:

The arguments are:

dayNum [optional]

A single *dayNum* argument is the number of a day within a month whose state should be turned *on*. If there are no arguments, none of the days in the month are turned on. The arguments can repeat any number of times, but each argument must be a whole number within the range of 1 to 32, inclusive.

Return value:

A new **`DayState`** object.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example constructs a day state for an application that displays the author's birth date in a *MonthCalendar* calendar control in bold. All other days of the year are displayed without emphasis.

```
::method getProperDayState private
  use strict arg date

  if date~orderedDate~substr(3, 2) == 07, date~orderedDate~right(2) == 14 then
    return .DayState~new(14)
  else
    return .DayState~new
```

16.39.3. value

```
>>--value----------------------------------------><
```

The *value* method returns the encoded value of the **`DayState`** object.

Arguments:

There are no arguments.

Return value:

The return is a number that encodes the state of every day within a month.

Remarks:

There is probably no practical use of this method for the programmer, other than perhaps curiosity. The method is used by the interpreter to get the encoded numerical value of a **`DayState`** object and send that number to the underlying month calendar control.

For those curious, the encoding is essentially a bit field. Each bit, 1 through 31 represents the state of the corresponding day in a month. If the bit is on, the month calendar displays that day in bold. If the bit is not on, the day is displayed normally.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example is for the curious and displays the bit encoding of a day state value:

```
dayState = .DayState~new(1, 4, 17)
say 'dayState value:' dayState~value 'binary form:' dayState~value~d2x~x2b

/* Output would be:

   dayState value: 65545 binary form: 00010000000000001001

*/
```

16.40. DayStates Class

A **DayStates** object is a sequential collection of day *DayState* objects. It is a speciality class used to construct the proper reply value for the *GETDAYSTATE* event of the *MonthCalendar* calendar control.

The primary purpose of the **DayStates** class is to supply the buffer of day states that is used to reply to the get day state event. But, the class also has methods that allow the programmer to build a cache of day state objects. Then during the get day state event, the programmer can request a buffer of some number of the cached day states.

16.40.1. Method Table

The following table lists the class and instance methods of the **DayStates** class:

Table 16.5. DayStates Class Method Reference

Method	Description
Class Methods	**Class Methods**
new	Instantiates a new **DayStates** object with a cache of **DayState** objects.
makeDayStateBuffer	Returns a buffer of the specified **DayState** objects
quickDayStateBuffer	Returns a buffer constructed from the 3 **DayState** objects specified.
Attributes	**Attributes**
endMonth	Reflects the last month in the cache of **DayState** objects.
startMonth	Reflects the first month in the cache of **DayState** objects.
Instance Methods	**Instance Methods**
putMonth	Puts a single **DayState** object in the cache.
putYear	Puts an entire year of **DayState** objects in the cache.
getDayState	Returns the specified **DayState** object from the cache.
getDayStateBuffer	Returns a day state buffer constructed from the specified months in the cache

16.40.2. makeDayStateBuffer (Class method)

```
>>--makeDayStateBuffer(--dayStates--)------------><
```

Returns a buffer of the specified day states. This buffer can be used for the return from the event handler for a *GETDAYSTATE* event.

Arguments:

The single argument is:

dayStates [required]

An array of *DayState* objects.

Return value:

The return is a day states buffer. The buffer is only useful as the reply to the GETDAYSTATE event of the month calendar control.

Remarks:

The *dayStates* array can not be sparse. The day state objects are assumed to be sequential by the underlying month calendar control.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example shows part of the implementation for a GETDAYSTATE event handler. Start with the month of the start date requested, and then, for the number of months requested, a day state object is created and put into an array. The *makeDayStateBuffer* method is invoked to get the proper buffer, which is returned from the event handler.

```
::method onGetDayState unguarded
  use arg startDate, count, id, hwnd

  -- Create the array to hold the .DayState objects.
  dayStates = .array~new(count)

  -- The calendar is restricted to a single year.  We
  -- know for our application, the month calendar always
  -- requests 3 months, the partial month prior to the
  -- current month showing, the current month, and the
  -- partial month following the current month.
  --
  -- So, if the starting month is December, it can only
  -- be the December prior to January of the current
  -- year.  It can not be the December of this year because
  -- the calendar will not display January of next year.
  -- Setting the month to 0 will produce a day state with
  -- no days turned on, exactly what we want.

  month = startDate~month
  if month == 12 then month = 0

  do i = 1 to count
    dayStates[i] = self~getDayState(month + i - 1)
  end

  buffer = .DayStates~makeDayStateBuffer(dayStates)
  return buffer


::method getDayState private
  use strict arg month

  select
    when month ==  1 then ds = .DayState~new(17)
    when month ==  2 then ds = .DayState~new(21)
    when month ==  3 then ds = .DayState~new
    ...
    otherwise ds = .DayState~new
  end
  -- End select
```

```
    return ds
```

16.40.3. new (Class Method)

```
>>--new(--+------------+--+---------+--)--------><
          +--startYear--+  +-,-count-+
```

Instantiates a new **DayStates** object and initializes the cache of *DayState* objects. Each day state in the is initialized with the state of every day turned off.

Arguments:

The arguments are:

startYear [optional]

Specifies the starting year for the cache. The default if this argument is omitted is two years prior to the current year. The *startYear* is specified as the whole number year value, e.g., 2011, 1988, 1492, etc..

count [optional]

The number of years to initialize the cache with. The default is 3 years.

Return value:

The return is a new **DayStates** object.

Remarks:

The cache of day state values in a day states object is always a whole number of years in size, and is always sequential. I.e., 12 day state values for each year. As indicated, initially the state of every day is turned off. The *putMonth* and *putYear* methods are used set the day state values in the cache to values with the state of days turned on.

The *makeDayStateBuffer* and *quickDayStateBuffer* class methods can be used to obtain a day states buffer without initializing or using a cache.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example instantiates a new **DayStates** object and initializes the cache to 10 years, with the start year one year prior to the current date. It then sets the state of July 4th of every year to on:

```
yr = (.DateTime~today~year - 1)

dayStates = .DayStates~new(yr, 10)

dayState = .DayState~new(4)
do 10
  dayStates~putMonth(yr, 7, dayState)
  yr += 1
end
```

16.40.4. quickDayStateBuffer (Class method)

```
>>--quickDayStateBuffer(--dayState1--,--dayState2--,--dayState3--)-------------><
```

Returns a buffer for the 3 specified day states. This buffer can be used for the return from the event handler for a *GETDAYSTATE* event.

Arguments:

The arguments are the 3 **`DayState`** objects used to construct the buffer.

dayState1 [required]

The first day state in the sequence of day states.

dayState2 [required]

The second day state in the sequence of day states.

dayState3 [required]

The third day state in the sequence of day states.

Return value:

The return is a day states buffer. The buffer is only useful as the reply to the GETDAYSTATE event of the month calendar control.

Remarks:

Normally the month calendar control is sized to only display 1 month. With this display, a week prior to the month and a week after the month are also shown. For this situation, when the underlying month calendar requests information on how to display individual days through the *GETDAYSTATE* notification, it will always request 3 months. The *quickDayStateBuffer* is a convenience method for this situation, which is the most common request.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example shows the event handler for the GETDAYSTATE notification in an application that shows the 1st and the 15th of each month in bold. The application always sizes the month calendar with 1 full month displayed. Because of this, we know the count will be 3.

```
::method onGetDayState unguarded
  use arg startDate, count, id, hwnd

  -- All months are the same, so we can ignore the start date.
  ds = .DayState~new(1, 15)
  return .DayStates~quickDayStateBuffer(ds, ds, ds)
```

16.40.5. endMonth (Attribute)

```
>>--endMonth-------------------------------------><
```

The *endMonth* attribute reflects the last month in the cache of *DayState* objects. Its value is a **`DateTime`** object whose date is the first of the last month and last year in the cache.

endMonth get:

Returns a **`DateTime`** object whose date is the last month in the cache.

endMonth set:

The *endMonth* attribute can not be set by the programmer. It is set internally by the class.

Remarks:

Since the cache always contains whole years, the date of the *endMonth* attribute is always going to be 12/01. The real information will be the year of the *endMonth* date. I.e., if the *startMonth* attribute is 1/1/1990 and the *endMonth* attribute is 12/01/2010 then the cache contains 21 complete years of **`DayState`** objects.

16.40.6. startMonth (Attribute)

```
>>--startMonth-----------------------------------><
```

The *startMonth* attribute reflects the first month in the cache of *DayState* objects. Its value is a **`DateTime`** object whose date is the first of January of the first year in the cache.

startMonth get:

Returns a **`DateTime`** object whose date is the first month in the cache.

startMonth set:

The programmer can not set the *startMonth* attribute. It is set internally by the class.

Remarks:

Since the cache always contains whole years, the date of the *startMonth* attribute is always going to be 1/1. The real information will be the year of the *startMonth* date. I.e., if the *startMonth* attribute is 1/1/2010 and the *endMonth* attribute is 12/01/2011 then the cache contains 2 complete years of **`DayState`** objects.

16.40.7. getDayState

```
>>--getDayState(--dateTime--)--------------------><
```

Retrieves the cached *DayState* object for the date specified.

Arguments:

The single argument is:

dateTime

A **`DateTime`** object that specifies which **`DayState`** object to retrieve. Only the month and the year of *dateTime* are relevant, the day is ignored.

Return value:

The value returned is the **`DayState`** object in the cache for the date specified, or **`.nil`** if there is no day state in the cache for the date.

Remarks:

Normally there is no need to get specific day state objects from the cache, the programmer usually wants a *buffer* of day state objects. However this method can be of use in debugging by checking what values are actually in the cache.

Example:

This example expands on the example for the *putYear* method by checking the values of the *startMonth* and *endMonth* day state objects.

```
  dayStates = .DayStates~new(2010, 3)
  dayState = .DayState~new(7, 14, 21, 28)
  do i = 2010 to 2012
    year = .DateTime~fromStandardDate(i || 0101)

    a = .array~new(12)
    do j = 1 to 12
      a[j] = dayState
    end

    dayStates~putYear(year, a)
  end

  s = dayStates~startMonth
  e = dayStates~endMonth

  say 'Start month day state value:' dayStates~getDayState(s)~value~d2x~x2b
  say 'End month day state value:  ' dayStates~getDayState(e)~value~d2x~x2b

::requires 'ooDialog.cls'

/* Output to the console would be:

Start month day state value: 1000000100000010000001000000
End month day state value:   1000000100000010000001000000

 */
```

16.40.8. getDayStateBuffer

```
>>--getDayStateBuffer(--dateTime--,--count--)----><
```

Creates a buffer from the cached *DayState* objects using the specified date and *count*. This buffer can be used for the return from the event handler for a *GETDAYSTATE* event.

Arguments:

The arguments are:

dateTime [required]

A **`DateTime`** object whose date is used as the starting point in the cache to fetch the **`DayState`** objects.

count [required]

The number of **`DayState`** objects to use in the returned buffer.

Return value:

The return is a day states buffer. The buffer is only useful as the reply to the GETDAYSTATE event of the month calendar control.

Remarks:

If some, or all, of the day state objects requested for the buffer are not in the cache, then a new **`DayState`** object is create with the start of all days turned off and used. The cache is not updated, it remains the same after the method returns.

Example:

This example sets up a cache of **`DayState`** objects before the application dialog is displayed. It then gets buffers from this cache during the *GETDAYSTATE* event to return from the event handler.

```
::class 'PaidHolidaysDlg' subclass ResDialog

::method init
  expose dayStates

  forward class (super) continue

  dayStates = self~createDayStateCache
  self~connectMonthCalendarEvent(IDC_MC_HOLIDAYS, "GETDAYSTATE", onGetDayState)
  ...

::method onGetDayState unguarded
  expose dayStates
  use arg startDate, count, id, hwnd

  return dayStates~getDayStateBuffer(startDate, count)
```

16.40.9. putMonth

```
>>--putMonth(--dateTime--,--dayState--)---------><
```

Puts a single *DayState* object into the cache.

Arguments:

The arguments are:

dateTime [required]

A **`DateTime`** object whose date specifies the month in the cache to put the *dayState*.

dayState [required]

The **`DayState`** object to put in the cache.

Return value:

There is no return value from this method.

Remarks:

If a day state object already exists for the month specified, it is replaced by *dayState*.

If the month specified by the *dateTime* argument does not yet exist in the cache, then the cache is extended to include the year specified by *dateTime*. The month specified is assigned the *dayState* argument and all other months in the extension are assigned a **`DayState`** object with a value of 0, (all days are turned off.)

Example:

This example shows how *putMonth* works. A **`DateTime`** object with the date of 7/1/2012 is used for the month and a **`DayState`** object is created with the 4th turned on. That will display July 4th 2012 bolded.

The output shows that before *putMonth* is invoked, the end month of the day states object was December 2011. After *putMonth* is invoked, the end month is December 2012. Adding July 2012 to the cache extended the entire year of 2012. All months other than July will have all days in the month turned off.

```
  dayStates = .DayStates~new(2010, 2)
  say "Day states end month:" dayStates~endMonth~standardDate

  month = .DateTime~fromStandardDate(20120701)
  dayState = .DayState~new(4)

  dayStates~putMonth(month, dayState)

  say "Day states end month:" dayStates~endMonth~standardDate

::requires "ooDialog.cls"

/* Output would be:

  Day states end month: 20111201
  Day states end month: 20121201

*/
```

16.40.10. putYear

```
>>--putYear(--dateTime--,--dayStates--)----------><
```

The *putYear* method adds a year of *DayState* objects to the cache.

Arguments:

The arguments are:

dateTime [required]

A **`DateTime`** object that specifies the year being added. Only the year portion of the date is relevant. The day and month of the date are ignored.

dayStates [required]

An **`array`** of **`DayState`** objects that are the day states for the months of the year being added. Only the indexes 1 through 12 are looked at. If the index contains a day state object, that day state object is assigned to the corresponding month. If an index does not have a value, then a day state object with a value of 0, (no days are turned on,) is assigned to the corresponding month.

Return value:

There is no return from this method.

Remarks:

If the year specified already exists in the cache then that year is replaced by the day state objects specified int the *dayStates* array.

If the year does not exist yet in the cache it is added. If adding the new year creates a gap in the sequential order of years in the cache, then the gap is filled in using years with the state of all days turned off. In other words, if the cache currently consists of the years 2008 through 2012 and the *putYear* method is used to put the year 2016 in the cache, then the cache is also filled with the years 2013 through 2015. Each year 2013 through 2015 have day state objects with the state of all days turned off.

Example:

This example creates a **`DayStates`** object with 3 years cached. In the application that uses this, all months have the 7th, 14th, 21st, and 28th days displayed in bold. After instantiating the new **`DayStates`** object, each year in the cache is updated with a **`DayState`** object that turns the state of the 7th, 14th, 21st, and 28th on.

```
  dayStates = .DayStates~new(2010, 3)
```

```
dayState = .DayState~new(7, 14, 21, 28)

do i = 2010 to 2012
  year = .DateTime~fromStandardDate(i || 0101)

  a = .array~new(12)
  do j = 1 to 12
    a[j] = dayState
  end

  dayStates~putYear(year, a)
end
```

Progress Bar Controls

A progress bar is a control that an application can use to indicate the progress of a lengthy operation. It consists of a rectangle that is gradually filled, from left to right or from bottom to top, with the system highlight color as an operation progresses. It has a range and a current position. The range represents the entire duration of the operation, and the current position represents the progress that the application has made toward completing the operation.

The underlying Windows progress bar control has been updated to allow a full 32-bit number to specify the minimum and maximum range and current position. Previously, the highest possible range or current position value was 65,535. The range is now -2147483648 through 2147483647.

Some sample programs included in the ooRexx distribution have examples of how to use the progress bar control. See for instance: **`oodPBar.rex`** and **`propDemo.rex`** in the **`samples\ooDialog`** directory.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with progress bar controls:

Instantiation:

Use the *newProgressBar*() method of the *dialog* to retrieve a progress bar object.

Dynamic Definition:

To dynamically add a progress bar to the dialog template of an *UserDialog* use the *createProgressBar*() method.

Event Notification

Unlike most dialog controls, the progress bar does not send any event notifications. Therefore there is no corresponding *connectProgressBarEvent* as there is for other dialog controls. For instance like the *connectDateTimePickerEvent* method of the *DateTimePicker* class.

17.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with ProgressBar objects, including the pertinent methods from other classes:

Table 17.1. ProgressBar Methods and Attributes

Method	Documentation
Useful External Methods	
newProgressBar	Obtains a ProgressBar object that represents a progress bar control in a dialog.
createProgressBar	Creates a ProgressBar control in the dialog template of an *UserDialog*.
Instance Methods	
backgroundColor	Sets the background color in the progress bar.
barColor	Sets the color of the progress indicator bar in the progress bar control.
getFullRange	Retrives the current high and low limits of the range of the progress bar.
getPos	Returns the current position of the progress bar.
setFullRange	Sets the minimum and maximum values for the progress bar using the full allowable range and returns the previous range.
setMarquee	Sets the progress bar to marquee mode. This causes the progress bar to move in a scrolling manner.

Method	Documentation
setPos	Sets the new position for a progress bar and redraws the bar to reflect the new position.
setStep	Specifies the step increment for the progress bar.
step	Advances the current position of the progress bar.

17.2. newProgressBar (dialog object method)

`ProgressBar` objects can not be instantiated by the programmer from Rexx code using a *new*() method. Rather, a date and time picker object is obtained by using the *newProgressBar*() method of the *dialog* object. The syntax is:

```
>>-newDateTimePicker(--id--)---------------------><
```

17.3. createProgressBar (UserDialog method)

A progress bar control can be added to the dialog template of a *UserDialog* through the *createProgressBar*() method. The basic syntax is:

```
>>--createProgressBar(--id--,--x--,--y--,--cx--,--cy-+-----------+--)----------><
                                                     +--,-style--+
```

17.4. backgroundColor

```
>>--backgroundColor(--colorRef--)----------------><
```

Sets the background color in the progress bar.

Arguments:

The required argument is:

colorRef

A COLORREF that specifies the new background color for the progress bar, or CLR_DEFAULT to set the background color back to its default color. If needed, use the *colorRef*() method of the *Image* class to construct the proper value for this argument.

Return value:

This method returns the previous background color, or CLR_DEFAULT if the progress bar was using the default color.

Details

This method is only effective in the Windows Classic theme.

Raises syntax errors when incorrect arguments are detected.

Example:

This example sets the background color to a custom color. This will only have effect in Windows Classic theme. It is not neccessary to set numeric digits to 11 to use this method. In the example this is only done to allow the return to be displayed in hexadecimal.

```
numeric digits 11
```

```
  progressBar = self~newProgressBar("IDC_PB_CONNECTING")

  say 'Going to set the background color for the progress bar'
  ret = progressBar~backgroundColor(.Image~colorRef(55, 111, 18))
  say 'Old background color was:' ret '(0x'ret~d2x')'

/* Output might be:

Going to set the background color for the progress bar
Old background color was: 4278190080 (0xFF000000)

*/
```

17.5. barColor

```
>>--barColor(--colorRef--)-----------------------><
```

Sets the color of the progress indicator bar in the progress bar control.

Arguments:

The required argument is:

colorRef

A COLORREF that specifies the new color for the bar in the progress bar, or CLR_DEFAULT to set the bar color back to its default color. If needed, use the *colorRef*() method of the *Image* class to construct the proper value for this argument.

Return value:

This method returns the previous bar color, or CLR_DEFAULT if the progress bar was using the default color.

Details

This method is only effective in the Windows Classic theme.

Raises syntax errors when incorrect arguments are detected.

Example:

This example sets the bar color to a custom color. This will only have effect in Windows Classic theme. It is not neccessary to set numeric digits to 11 to use this method. In the example this is only done to allow the return to be displayed in hexadecimal.

```
  numeric digits 11

  progressBar = self~newProgressBar("IDC_PB_CONNECTING")

  say 'Going to set the bar color for the progress bar'
  ret = progressBar~barColor(.Image~colorRef(180, 255, 110))
  say 'Old bar color was:       ' ret  '(0x'ret~d2x')'

/* Output might be:

Going to set the bar color for the progress bar
Old bar color was:        4278190080 (0xFF000000)

*/
```

17.6. getFullRange

```
>>--getFullRange---------------------------------><
```

Retrives the current high and low limits of the range of the progress bar.

Arguments:

There are no arguments to this method.

Return value:

The range is returned in a **`.Directory`** object with the following indexes:

1. MIN - The current minimum limit, as a signed number, for the progress bar.

2. MAX - The current maximum limit, as a signed number, for the progress bar.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example demonstrates how the range is returned. In the resetRange() method a directory object is passed in. For this example, assume range~min is -200 and range~max is 200.

```
::method resetRange
  use strict arg range
  pb = self~newProgressBar("IDC_PB_COUNTDOWN")
  if pb \= .nil then do
     pb~setRange(range~min, range~max)
     pb~setStep(10)
     pb~setPos(-200)
  end
  ...

::method displayCurrentRange private
  expose pb

  range = pb~getRange
  say 'Current minimum:' range~min 'current maximum:' range~max
  say
  say 'Current minimum:' pb~getRange~min 'current maximum:' pb~getRange~max

/* Output would be:

Current minimum: -200 current maximum: 200

Current minimum: -200 current maximum: 200

*/
```

17.7. getPos

```
>>--getPos---------------------------------------><
```

This method returns the current position of the progress bar.

Arguments:

There are no arguments.

Return value:

The current position of the progress bar is returned.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

In this example, the range for the progress bar is set to -200 to 200. At another point in the program, the current position is printed to the screen. This is not real-world use case, just an example:

```
  pb = self~newProgressBar("IDC_PB_COUNTDOWN")
  if pb \= .nil then do
     pb~setFullRange(-200,200)
     pb~setStep(10)
     pb~setPos(-200)
  end
  ...

::method display private
  expose pb

  pb~step
  say 'Now at position' pb~getPos

/* Output might be:

Now at position -190
Now at position -180
Now at position -170
Now at position -160
...

*/
```

17.8. setFullRange

```
Form 1:

>>--setRange(--+------------------+--)-----------><
               +-aDirectoryObject-+

Form 2:

>>--setRange(--+-----+--,--+-----+--)------------><
               +-min-+     +-max-+
```

The *setFullRange*() method sets the minimum and maximum values for the progress bar using the full allowable range and returns the previous range. The minimum and maximum values are signed numbers. To use the entire range possible the programmer would use -2147483648 through 2147483647 for the minimum and maximum.

Arguments:

- Form 1:

  aDirectoryObject

  Optional. If omitted, the minimum range value is set to 0 and the maximum to 100. Otherwise the following indexes of the directory are used to set the range:

1. MIN the minimum range value. 0 is the default if the index is not present.

2. MAX the maximum range value. 100 is the default if the index is not present.

- Form 2:
  min

  Optional. The minimum range value. The default is 0.

  max

  Optional. The maximum range value. The default is 100.

Return value:

A directory object containing the previous range for the progress bar. The index MIN contains the previous minimum range value and the index MAX contains the previous maximum. The returned values use the full allowable range of the progress bar and are not restricted to 0 though 65535

Details

Raises syntax errors when incorrect arguments are detected.

17.9. setMarquee

```
>>--setMarquee(--+------+--+-----------+--)------><
                 +--on--+  +--,-pause--+
```

Sets the progress bar to marquee mode. This causes the progress bar to move in a scrolling manner. Use this method when you do not know the amount of progress toward completion but wish to indicate that progress is being made.

This method is used to start or stop the scrolling. Note that the progress bar has to have the PBS_MARQUEE style. When using the *createProgressBar*() method in a *UserDialog* class the style keyword is MARQUEE. Otherwise, a resource editor will handle adding the PBS_MARQUEE style flag to the resource script.

Each time the marquee mode is turned on, the progress bar is reset to the minimum of its range.

Arguments:

The arguments are:

on

Optional. True or false. True to start the marquee, false to stop it. The default is true to start the marquee.

pause

Optional. The time in milliseconds between the animation. The default is 1000 (one second.)

Return value:

This method always returns `.true`.

Details

This method requires Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version* method to determine the current version of the library.

Raises syntax errors when incorrect arguments are detected.

Requires the progress bar to have the PBS_MARQUEE style.

Example:

This example turns marquee mode on and sets the time between updates at one half of a second:

```
pb = self~newProgressBar("IDC_PB_SEARCHING")
if pb \== .nil then pb~setMarquee(.true, 500)
```

17.10. setPos

```
>>--setPos(--newPos--)---------------------------><
```

The setPos method sets the new position for a progress bar and redraws the bar to reflect the new position.

Arguments:

The single required argument is the new position for the progress bar.

Return value:

The previous position is returned.

Details

Raises syntax errors when incorrect arguments are detected.

17.11. setStep

```
>>--setStep(--+---------+--)---------------------><
              +-newstep-+
```

This method specifies the step increment for the progress bar. The step increment is the amount by which the progress bar advances its current position when the *step*() method is called without an increment value.

By default, the underlying progress bar uses a default step of 10. It is not neccessary to set the step at all if 10 is satisfactory.

Arguments:

The single optional argument is the new step increment. The default step increment is 10.

Return value:

This method returns the previous step increment.

Details

Raises syntax errors when incorrect arguments are detected.

17.12. step

```
>>--step(--+-----------+--)----------------------><
           +-increment-+
```

The step method advances the current position of the progress bar. If *increment* is specified, the current position is incremented by that amount. If not specified, the current position is incremented by the amount set with the *setStep*() method. The progress bar is redrawn to reflect the new position.

There is this important difference between using an increment, usually called a delta, and stepping by the amount specified in the setStep() method. When an increment is used and the increment would position the progress bar past its maximum range, the progress bar is advanced to the end and stays there.

On the other hand, when the default step is used and the progress bar reaches the end of its range, it is reset to the minimum.

Details

Raises syntax errors when incorrect arguments are detected.

Arguments:

The single optional argument is the amount to advance the current position. If this argument is not used, the position is advanced by the value set with the setStep method.

Return value:

The previous position.

ReBar Controls

xxx

yyy

The **`ReBar`** class provides methods to work with and manipulate the underlying Windows rebar dialog control which it represents. It is a concrete subclass of the dialog *control* object and therefore has all methods of the dialog control object.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with rebar controls:

Instantiation:

Use the *newReBar* method of the *dialog* object to retrieve a new ReBar object.

Dynamic Definition:

To dynamically define a rebar in a *UserDialog* class, use the *createReBar* method.

Event Notification

To connect the *event* notifications sent by the underlying rebar control to a method in the Rexx dialog object use the *connectReBarEvent* method.

Window Creation:

To dynamically create a rebar window in a dialog, use the *createReBarWindow* method. This can only be done after the underlying Windows dialog has been created.

18.1. Method Table

The following table provides links to the documentation for the primary classes, methods, and attributes used in working with ReBar objects, including the pertinent classes, and methods from other classes:

Table 18.1. Important Tab Methods

Method	Description
Useful	**Classes**
ReBarBandInfo	**`ReBarBandInfo`**objects xxx.
Useful	**External Methods**
newReBar	Returns a **`ReBar`** object for the control with the specified ID.
createReBar	Creates a rebar control in the dialog template of a *UserDialog*
createReBarWindow	Creates a rebar control in an existing Windows dialog.
connectReBarEvent	Connects rebar event notifications to a method in the Rexx dialog object
Instance Methods	
deleteBand	x.

18.2. newReBar (dialog object method)

ReBar objects can not be instantiated by the programmer from Rexx code. Rather a ReBar object is obtained by using the *newReBar*() method of the *dialog* object. The syntax is:

```
>>-newReBar(--id--)---------------------><
```

18.3. createReBar (UserDialog method)

A rebar control can be added to the dialog template for a *UserDialog* dialog through the *createReBar*() method. The basic syntax is:

```
>>--createReBar(-id-,--x-,--y-,--cx-,--cy-+---------+--+----------------+--)--><
                                          +-,-style-+  +-,-attributeName-+
```

18.4. createReBarWindow (CreateWindows method)

A rebar control window can be created and added to any dialog through the *createReBarWindow*() method. The basic syntax is:

```
>>--createReBarWindow(--id--+----------+--)------><
                            +-,-style--+
```

18.5. connectReBarEvent (dialog object method)

To connect event notifications from a rebar control use the *connectReBarEvent*() method of the *dialog* object. The basic syntax is:

```
>>-connectReBarEvent(--id--,--event--+---------------+--+-------------+--)----><
                                     +--,-methodName--+  +-,-willReply--+
```

18.6. deleteBand

```
>>--deleteBand(--+--------+--)--------------------------------------------><
                 +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

18.7. getBandBorders

```
>>--getBandBorders(--+--------+--)---------------------------------------------><
                     +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

18.8. getBandCount

```
>>--getBandCount(--+--------+--)---------------------------------------------><
                     +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.9. getBandInfo

```
>>--getBandInfo(--+--------+--)--------------------------------------------><
                  +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.10. getBandMargins

```
>>--getBandMargins(--+--------+--)-----------------------------------------><
                  +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.11. getBarHeight

```
>>--getBarHeight(--+--------+--)--------------------------------------------><
                   +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.12. getBarInfo

```
>>--getBarInfo(--+--------+--)--------------------------------------------><
                 +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:
: xx

Remarks:
: Additional comments.

Details
: Raises syntax errors when incorrect usage is detected.

  Sets the *.SystemErrorCode* variable.

Example:
: This example ...

```
```

18.13. getBkColor

```
>>--getBkColor(--+--------+--)------------------------------------------------><
                 +--type--+
```

xx

Arguments:
: The arguments are:

  TERM
  : xx

Return value:
: xx

Remarks:
: Additional comments.

Details
: Raises syntax errors when incorrect usage is detected.

  Sets the *.SystemErrorCode* variable.

Example:
: This example ...

```
```

18.14. getColorScheme

```
>>--getColorScheme(--+--------+--)--------------------------------------------><
                 +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.15. getExtendedStyle

```
>>--getExtendedStyle(--+--------+--)---------------------------------------------><
                       +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.16. getImageList

```
>>--getImageList(--+--------+--)--------------------------------------------><
                   +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.17. getPalette

```
>>--getPalette(--+--------+--)----------------------------------------------><
                 +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

18.18. getRect

```
>>--getRect(--+--------+--)---------------------------------------------><
              +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

18.19. getRowCount

```
>>--getRowCount(--+--------+--)---------------------------------------------><
                  +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

18.20. getRowHeight

```
>>--getRowHeight(--+--------+--)---------------------------------------------><
                   +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

18.21. getTextColor

```
>>--getTextColor(--+--------+--)---------------------------------------------><
                   +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.22. getToolTips

```
>>--getToolTips(--+--------+--)---------------------------------------------><
                  +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.23. hitTestInfo

```
>>--hitTestInfo(--+--------+--)---------------------------------------------><
                  +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.24. idToIndex

```
>>--idToIndex(--+--------+--)--------------------------------------------><
                +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.25. insertBand

```
>>--insertBand(--+--------+--)---------------------------------------------><
                 +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.26. maximizeBand

```
>>--maximizeBand(--+--------+--)-------------------------------------------><
                   +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.27. minimizeBand

```
>>--minimizeBand(--+--------+--)---------------------------------------------><
                   +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.28. moveBand

```
>>--moveBand(--+--------+--)---------------------------------------------><
               +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.29. pushChevron

```
>>--pushChevron(--+--------+--)--------------------------------------------><
                  +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.30. setBandInfo

```
>>--setBandInfo(--+--------+--)--------------------------------------------><
                  +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

18.31. setBandWidth

```
>>--setBandWidth(--+--------+--)---------------------------------------------><
                   +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

18.32. setBarInfo

```
>>--setBarInfo(--+--------+--)---------------------------------------------><
                   +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.33. setBkColor

```
>>--setBkColor(--+--------+--)--------------------------------------------><
                  +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.34. setColorScheme

```
>>--setColorScheme(--+--------+--)----------------------------------------><
                  +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

18.35. setExtendedStyle

```
>>--setExtendedStyle(--+--------+--)------------------------------------------><
                       +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

18.36. setImageList

```
>>--setImageList(--+--------+--)--------------------------------------------><
                   +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.37. setPalette

```
>>--setPalette(--+--------+--)----------------------------------------------><
                 +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.38. setParent

```
>>--setParent(--+--------+--)---------------------------------------------><
                +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.39. setTextColor

```
>>--setTextColor(--+--------+--)------------------------------------------><
                   +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

18.40. setToolTips

```
>>--setToolTips(--+--------+--)---------------------------------------------><
                  +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

18.41. setWindowTheme

```
>>--setWindowTheme(--+--------+--)------------------------------------------><
                  +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.42. showBand

```
>>--showBand(--+--------+--)-------------------------------------------><
                +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.43. sizeToRect

```
>>--sizeToRect(--+--------+--)-------------------------------------------><
                  +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

18.44. ReBarBandInfo Class

A *ReBarBandInfo* object represents a band in a *ReBar*. The attributes of a *ReBarBandInfo* object define a band in a rebar. A rebar band info object is also used to retrieve information about a band.

xx

18.44.1. Method Table

The following table lists the class and instance methods of the **`ReBarBandInfo`** class:

Table 18.2. ReBarBandInfo Class Method Reference

Method	Description
Class Methods	**Class Methods**
new	Instantiates a new **`ReBarBandInfo`** object.
Attribute Methods	**Attribute Methods**
bitmapBack	xx.
chevronLocation	xx.
chevronState	xx.
child	xx.
clrBack	xx.
clrFore	xx.

18.44.2. new (Class Method)

```
>>--new(--+--------+--)-------------------------------------------><
          +--type--+
```

xx

Arguments:

xx

TERM
    xx

Return value:
    xx

Remarks:
    Additional comments.

Details:
    Raises syntax errors when incorrect usage is detected.

    Sets the *.SystemErrorCode* variable.

Example:
    This example ...

```
```

18.44.3. bitmapBack (Attribute)

```
>>--bitmapBack-----------------------------------------------------><

>>--bitmapBack = varName-------------------------------------------><
```

xx

bitmapBack get:
    details about get

bitmapBack set:
    details about set

Remarks:
    Additional comments.

Details
    Raises syntax errors when incorrect usage is detected.

Example:
    This example ...

```
```

18.44.4. chevronLocation (Attribute)

```
>>--chevronLocation-----------------------------------------------------><

>>--chevronLocation = varName-------------------------------------------><
```

xx

chevronLocation get:
    details about get

chevronLocation set:
details about set

Remarks:
Additional comments.

Details
Raises syntax errors when incorrect usage is detected.

Example:
This example ...

```

```

18.44.5. chevronState (Attribute)

```
>>--chevronState-----------------------------------------------------><

>>--chevronState = varName-------------------------------------------><
```

xx

chevronState get:
details about get

chevronState set:
details about set

Remarks:
Additional comments.

Details
Raises syntax errors when incorrect usage is detected.

Example:
This example ...

```

```

18.44.6. child (Attribute)

```
>>--child------------------------------------------------------------><

>>--child = varName--------------------------------------------------><
```

xx

child get:
details about get

child set:
details about set

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example ...

```

```

18.44.7. clrBack (Attribute)

```
>>--clrBack-----------------------------------------------------><

>>--clrBack = varName-------------------------------------------><
```

xx

clrBack get:

details about get

clrBack set:

details about set

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example ...

```

```

18.44.8. clrFore (Attribute)

```
>>--clrFore-----------------------------------------------------><

>>--clrFore = varName-------------------------------------------><
```

xx

clrFore get:

details about get

clrFore set:

details about set

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example ...

```
```

18.44.9. cx (Attribute)

```
>>--cx-----------------------------------------------------><

>>--cx = varName-------------------------------------------><
```

xx

cx get:

details about get

cx set:

details about set

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example ...

```
```

18.44.10. cxHeader (Attribute)

```
>>--cxHeader-----------------------------------------------------><

>>--cxHeader = varName-------------------------------------------><
```

xx

cxHeader get:

details about get

cxHeader set:

details about set

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example ...

```
```

18.44.11. cxIdeal (Attribute)

```
>>--cxIdeal-----------------------------------------------------><

>>--cxIdeal = varName-------------------------------------------><
```

xx

cxIdeal get:

details about get

cxIdeal set:

details about set

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example ...

```
```

18.44.12. cxMinChild (Attribute)

```
>>--cxMinChild--------------------------------------------------><

>>--cxMinChild = varName----------------------------------------><
```

xx

cxMinChild get:

details about get

cxMinChild set:

details about set

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example ...

```
```

18.44.13. cyChild (Attribute)

```
>>--cyChild-----------------------------------------------------><

>>--cyChild = varName-------------------------------------------><
```

xx

cyChild get:
: details about get

cyChild set:
: details about set

Remarks:
: Additional comments.

Details
: Raises syntax errors when incorrect usage is detected.

Example:
: This example ...

```
```

18.44.14. cyIntegral (Attribute)

```
>>--cyIntegral-----------------------------------------------------><

>>--cyIntegral = varName-------------------------------------------><
```

xx

cyIntegral get:
: details about get

cyIntegral set:
: details about set

Remarks:
: Additional comments.

Details
: Raises syntax errors when incorrect usage is detected.

Example:
: This example ...

```
```

18.44.15. cyMaxChild (Attribute)

```
>>--cyMaxChild-----------------------------------------------------><

>>--cyMaxChild = varName-------------------------------------------><
```

xx

cyMaxChild get:
details about get

cyMaxChild set:
details about set

Remarks:
Additional comments.

Details
Raises syntax errors when incorrect usage is detected.

Example:
This example ...

```

```

18.44.16. cyMinChild (Attribute)

```
>>--cyMinChild-----------------------------------------------------><

>>--cyMinChild = varName-------------------------------------------><
```

xx

cyMinChild get:
details about get

cyMinChild set:
details about set

Remarks:
Additional comments.

Details
Raises syntax errors when incorrect usage is detected.

Example:
This example ...

```

```

18.44.17. id (Attribute)

```
>>--id-------------------------------------------------------------><
```

```
>>--id = varName-------------------------------------------><
```

xx

id get:
  details about get

id set:
  details about set

Remarks:
  Additional comments.

Details
  Raises syntax errors when incorrect usage is detected.

Example:
  This example ...

```

```

18.44.18. imageIndex (Attribute)

```
>>--imageIndex-----------------------------------------------------><

>>--imageIndex = varName-------------------------------------------><
```

xx

imageIndex get:
  details about get

imageIndex set:
  details about set

Remarks:
  Additional comments.

Details
  Raises syntax errors when incorrect usage is detected.

Example:
  This example ...

```

```

18.44.19. itemData (Attribute)

```
>>--itemData-----------------------------------------------------><

>>--itemData = varName-------------------------------------------><
```

xx

itemData get:
details about get

itemData set:
details about set

Remarks:
Additional comments.

Details
Raises syntax errors when incorrect usage is detected.

Example:
This example ...

```
```

18.44.20. mask (Attribute)

```
>>--mask-----------------------------------------------------><

>>--mask = varName-------------------------------------------><
```

xx

mask get:
details about get

mask set:
details about set

Remarks:
Additional comments.

Details
Raises syntax errors when incorrect usage is detected.

Example:
This example ...

```
```

18.44.21. style (Attribute)

```
>>--style----------------------------------------------------><

>>--style = varName------------------------------------------><
```

xx

style get:
details about get

style set:
    details about set

Remarks:
    Additional comments.

Details
    Raises syntax errors when incorrect usage is detected.

Example:
    This example ...

```

```

18.44.22. text (Attribute)

```
>>--text-----------------------------------------------------><

>>--text = varName-------------------------------------------><
```

xx

text get:
    details about get

text set:
    details about set

Remarks:
    Additional comments.

Details
    Raises syntax errors when incorrect usage is detected.

Example:
    This example ...

```

```

Scroll Bar Controls

A scroll bar is a dialog control that allows a user to scroll data like continuous text or images in a window in order to bring into view the portions of the data that extend beyond the borders of the window. The bar that is movable in the scroll bar is called the "scroll box".

The **`ScrollBar`** class provides methods to query and modify scroll bar controls.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with scroll bar controls:

Instantiation:

Use the *newScrollBar*() method of the *dialog* object to retrieve a scroll bar object.

Dynamic Definition:

To dynamically create a scroll bar in the dialog template of a *UserDialog* use the *createScrollBar* method.

Event Notification

To connect the *event* notifications sent by the underlying scroll bar control to a method in the Rexx dialog object use the *connectScrollBarEvent*) method.

19.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with ScrollBar objects, including the pertinent methods from other classes:

Table 19.1. ScrollBar Methods and Attributes

Method	Documentation
Useful External Methods	
newScrollBar	Obtains a ScrollBar object that represents a scroll bar control in a dialog.
createScrollBar	Creates a scroll bar control in an *UserDialog*.
connectScrollBarEvent	Connects a single scroll bar event notification to a Rexx dialog method.
connectAllSBEvents	Connects all scroll bar event notifications to a single Rexx dialog method.
connectEachSBEvent	Connects a number of scroll bar event notifications, each to its own Rexx dialog method, in a single method invocation.
Constants	**Constants**
Scroll Event Constants	The **`ScrollBar`** class provides a set of ::constant values for the numeric scroll event codes.
Instance Methods	
determinePosition	Determines the new position of the scroll box based on the position sent with the scroll bar event notification message.
position	Retrieves the position of the scroll box in the associated scroll bar.
range	Retrieves the minimum and maximum positions of the associated scroll bar.
setRange	Sets the minimum and maximum positions for the associated scroll bar.
setPos	Sets the position of the scroll box for the associated scroll bar.

19.2. newScrollBar (dialog object method)

`ScrollBar` objects can not be instantiated by the programmer from Rexx code using a *new*() method. Rather, a scroll bar object is obtained by using the *newScrollBar*() method of the *dialog* object. The syntax is:

```
>>--newScrollBar(--id--)-------------------------><
```

19.3. createScrollBar (UserDialog method)

A scroll bar control can be created in the dialog template of a *UserDialog* through the *createScrollBar* method. The basic syntax is:

```
>>--createScrollBar(-id-,--x-,--y-,--cx-,--cy-+---------+-+------------+--)--><
                                              +-,--style-+ +-,--attrName-+
```

19.4. connectScrollBarEvent (dialog object method)

To connect an individual event notification from a scroll bar control use the *connectScrollBarEvent* method of the *dialog* object. The basic syntax is:

```
>>--connectScrollBarEvent(--id--,--event--+---------------+-+-------------+-)--><
                                          +--,-methodName--+ +-,-willReply--+
```

19.5. connectAllSBEvents (dialog object method)

To connect all event notifications from a scroll bar control to a single method use the *connectAllSBEvents* method of the *dialog* object. The basic syntax is:

```
>>--connectAllSBEvents(-id-,-mName-+------+-+------+-+-----+-+-------------+-)--><
                                   +-,-mn-+ +-,-mx-+ +-,-p-+ +-,-willReply--+
```

19.6. connectEachSBEvent (dialog object method)

To connect a number of event notifications from a scroll bar control, and specify an individual method for each event, in one method invocation use the *connectEachSBEvent* method of the *dialog* object. The basic syntax is:

```
>>--connectEachSBEvent(-id-,-mthWhenUp-,-mthWhenDown-+---------------+--------->
                                                     +-,-mthWhenDrag-+

>--+-------+-+-------+-+-------+--+-----------+-+-----------+-+----------+----->
   +-,-min-+-+-,-max-+ +-,-pos-+  +-,-mthPgUp-+ +-,-mthPgDn-+ +-,-mthTop-+

>--+-------------+-+------------+-+------------+-+-------------+--)------------><
   +-,-mthButtom-+ +-,-mthTrack-+ +-,-mthEndSc-+ +-,-willReply-+
```

19.7. Scroll Event Constants

When a scroll bar sends an *event* notification *message*, one of the parameters contains the numeric code, (defined by Windows,) for the event. The **`ScrollBar`** class provides **`::constant`** values for each of the event codes. This allows the programmer to determine which event is represented by the event code without having to know what the individual numeric value is for each code.

Connecting the scroll bar event notifications to a method in the Rexx dialog object is done through the *connectScrollBarEvent* method. The second argument to that method is the event *keyword* that is to be connected. Do not confuse the keyword and the constant. The event *keyword* is a string that identifies the event to be connected to the ooDialog framework. The event *constant* is a numeric value that Windows sends in the notification message to identify the event. The following table lists the constant name, its matching keyword in the *connectScrollBarEvent* method, and a description of the event referred to:

Table 19.2. ScrollBar Event Constants

Constant	KeyWord	Event
LINEUP	LineUp	The scroll bar was scrolled up by one unit.
LINELEFT	LineLeft	The scroll bar was scrolled to the left by one unit.
LINEDOWN	LineDown	The scroll bar was scrolled down by one unit.
LINERIGHT	LineRight	The scroll bar was scrolled to the right by one unit.
PAGEUP	PageUp	The scroll bar was scrolled up by one page size.
PAGELEFT	PageLeft	The scroll bar was scrolled to the left by one page size.
PAGEDOWN	PageDown	The scroll bar was scrolled down by one page size.
PAGERIGHT	PageRight	The scroll bar was scrolled to the right by one page size.
THUMBPOSITION	Position	The scroll box (thumb) of the scroll bar was dragged and the user has released the mouse button.
THUMBTRACK	Drag	The user is dragging the scroll box. This message is sent repeatedly until the user releases the mouse button.
TOP	Top	The scroll bar was scrolled completely to the top.
LEFT	Left	The scroll bar was scrolled completely to the left.
BOTTOM	Bottom	The scroll bar was scrolled completely to the bottom.
RIGHT	Right	The scroll bar was scrolled completely to the right.
ENDSCROLL	EndScroll	Scrolling has been ended, that is, the appropriate key or mouse button has been released.

19.8. determinePosition

```
>>--determinePosition(--posData--,--+------------+--+-------------+--)--------><
                                    +-singleStep-+  +-,--pageStep-+
```

The *determinePosition* method determines the new position of the scroll box based on the position info sent with the scroll bar notification messages.

Arguments:

The arguments are:

posData [required]

The position information sent with the scroll bar notification messages.

singleStep [optional]

The value by which the scroll box position is increased or decreased when the user performs a single-step event like using the Down or Up arrow keys or clicking on the arrow buttons of the scroll bar. The default if this argument is omitted is 1.

pageStep [optional]

The value by which the scroll box position is increased or decreased when the user performs a page-step event like using the PgDn or PgUp arrow keys or clicking on an area in the scroll bar that is not occupied by the scroll box or the arrow buttons. The default if this argument is omitted is 10.

Return value:

The resulting position based on *posData* and the current position.

Note

The position of a scroll bar cannot be modified directly by a user. The ooDialog program must react to the notification that is the result of the user interaction and set the resulting position of the scroll box using *setPos*. Use the *determinePosition* method to determine the resulting position within your notification handler for the scroll bar notification messages.

19.9. position

```
>>--position-------------------------------------><
```

The position method retrieves the position of the scroll box in the associated scroll bar.

Return value:

The position of the scroll box.

19.10. range

```
>>--range----------------------------------------><
```

The range method retrieves the minimum and maximum positions of the associated scroll bar.

Return value:

The minimum and maximum positions, separated by a blank.

19.11. setPos

```
>>--setPos(--position--+-----------+--)----------><
                       +-,--redraw-+
```

The setPos method sets the position of the scroll box for the associated scroll bar.

Arguments:

The arguments are:

position

The position to which the scroll box is to be moved.

redraw

If you specify 1 or Y, the scroll bar is redrawn using the new position. Otherwise, the new position is set, but the scroll bar display is not updated. The default value is 1.

Return value:

0

Setting the position was successful.

1

For all other cases.

19.12. setRange

```
>>--setRange(--min--,--max--+-----------+--)-----><
                            +-,--redraw-+
```

The setRange method sets the minimum and maximum positions for the associated scroll bar.

Arguments:

The arguments are:

min

The minimum position to which the scroll bar can be moved.

max

The maximum position to which the scroll bar can be moved.

redraw

If you specify 1 or Y, the scroll bar is redrawn using the new range. Otherwise, the range is set but the scroll bar display is not updated. The default value is 1.

Return value:

0

Setting the range was successful.

1

For all other cases.

Static Controls

Static controls are used in applications as labels or to separate groups of controls. Although static controls are child windows, they cannot be selected. Because of that, they do not receive the keyboard focus and do not have a keyboard interface. Since static controls are not meant to interact with the user, the **`Static`** class has relatively few methods, compared to some of the other dialog control classes.

A static control can have a notify style. With this style the control will receive mouse input and then notifies its parent dialog when the user clicks or double clicks the control.

There are four basic types of static controls. Each type has one or more styles.
Simple Graphics
Text
Image
Owner-Drawn

Simple graphic static controls draw a filled rectangle, a frame, or an etched line. They are primarily used to separate groups of controls. Text static controls display text. They are usually used for labels, or to display information to the user. Image static controls display an image, icon, bitmap or enhanced metafile. ooDialog does not have much, if any, support for owner-drawn static controls.

The **`Static`** class provides methods to work with and manipulate the underlying Windows static dialog control which it represents. It is a concrete subclass of the dialog *control* object and therefore has all methods of the of the dialog control object.

By default, static controls are assigned a resource ID of -1. There is no way to query or modify static controls unless the control has a positive resource ID. Which in most circumstances is fine, usually static controls do not need to be modified. Be sure to assign a positive resource ID to static controls that will be modified during the execution of a dialog.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with static controls:

Instantiation:

Use the *newStatic* method of the *dialog* object to retrieve a new Static object.

Dynamic Definition:

To dynamically define a static control in a *UserDialog* class, use one of the methods listed in the Create *Static* Controls section of the **`UserDialog`** documentation. There are a number of convenience methods for creating different types of static controls.

Event Notification

To connect the *event* notifications sent by the underlying static control to a method in the Rexx dialog object use the *connectStaticEvent*() method.

20.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with Static objects, including the relevant methods from other classes:

Table 20.1. Important Static Methods

Method	Description
Useful	**External Methods**
newStatic	Returns a **`Static`** object for the control with the specified ID.

Method	Description
createBlackFrame	Creates a static black frame control in the dialog template.
createBlackRect	Creates a static black rectangle control in the dialog template.
createEtchedFrame	Creates a static etched frame control in the dialog template.
createEtchedHorizontal	Creates a static control in the dialog template that is an etched horizontal line.
createEtchedVertical	Creates a static frame control in the dialog template that is a single etched vertical line.
createGrayFrame	Creates a static gray frame control in the dialog template.
createGrayRect	Creates a static gray rectangle control in the dialog template.
createStatic	Creates a static control in the dialog template
createStaticText	Creates a static text control in the dialog template.
createStaticImage	Creates a static image control in the dialog template.
createWhiteFrame	Creates a white static frame control in the dialog template.
createWhiteRect	Creates a white static rectangle control in the dialog template.
connectStaticEvent	Connects static event notifications to a method in the Rexx dialog object
Instance Methods	**Instance Methods**
getIcon	Gets the static control's icon image, if there is one.
getImage	Returns the static control's image, if there is one.
setIcon	Sets or removes the icon image for the static control.
setImage	Sets or removes the image for the static control.

20.2. createBlackFrame (UserDialog method)

The *createBlackFrame* method creates a black *frame* static control in the dialog template. The rectangle is drawn with the current window frame color, which is black in the default color scheme. This is the basic syntax:

```
>>--createBlackFrame(--+------+--x-,-y-,-cx-,-cy--+-----------+-)--------------><
                       +--id--+                   +-,--style--+
```

20.3. createBlackRect (UserDialog method)

The *createBlackRect* method creates a black *rectangle* static control in the dialog template. The rectangle is filled with the current window frame color, which is black in the default color scheme. This is the basic syntax:

```
>>--createBlackRect(--+------+--x-,-y-,-cx-,-cy--+-----------+-)---------------><
                      +--id--+                   +-,--style--+
```

20.4. createEtchedFrame (UserDialog method)

The *createEtchedFrame* method adds an etched *frame* static control to the dialog template. This is the basic syntax:

```
>>--createEtchedFrame(--+-------+--x-,-y-,-cx-,-cy--+-----------+-)------------><
```

```
                          +--id-,-+                    +-,--style--+
```

20.5. createEtchedHorizontal (UserDialog method)

The *createEtchedHorizontal* method adds an etched horizontal *line* static control to the dialog.

```
>>--createEtchedHorizontal(--+-------+--x-,-y-,-cx-,-cy--+-----------+-)------><
                             +--id-,-+                   +-,--style--+
```

20.6. createEtchedVertical (UserDialog method)

The *createEtchedVertical* method adds an etched vertical *line* static control to the dialog.

```
>>--createEtchedVertical(--+-------+--x-,-y-,-cx-,-cy--+-----------+-)--------><
                           +--id-,-+                   +-,--style--+
```

20.7. createGrayFrame (UserDialog method)

The *createGrayFrame* method creates a gray *frame* static control in the dialog template. The frame is drawn with the same color as the screen background (desktop), which is gray in the default color scheme. This is the basic syntax:

```
>>--createGrayFrame(--+------+--x-,-y-,-cx-,-cy--+-----------+-)--------------><
                      +--id--+                   +-,--style--+
```

20.8. createGrayRect (UserDialog method)

The *createGrayRect* method creates a gray *rectangle* static control in the dialog template. The rectangle is filled with the same color as the screen background (desktop), which is gray in the default color scheme. This is the basic syntax:

```
>>--createGrayRect(--+------+--x-,-y-,-cx-,-cy--+-----------+-)---------------><
                     +--id--+                   +-,--style--+
```

20.9. createStatic (UserDialog method)

A static control can be added to the dialog template for a *UserDialog* dialog through the *createStatic*() method. This method uses the *style* argument to specify any type and style of static control. The basic syntax is:

```
>>--createStatic(--+----+--,-x-,-y-,-cx-,-cy--+----------+--+---------+--)-----><
                   +-id-+                     +-,-style--+  +-,-text--+
```

20.10. createStaticImage (UserDialog method)

The *createStaticImage* method creates a static control that displays images in the dialog template. The basic syntax is:

```
>>--createStaticImage(--id--,-x--,-y--,-cx--,-cy--+----------+--)-------------><
                                                  +-,-style--+
```

20.11. createStaticText (UserDialog method)

The *createStaticText* method creates a static text control in the dialog template. The basic syntax is:

```
>>-createStaticText(-+------+--x-,--y-+------+-+------+-+----------+-+--------+-)-><
                     +-id-,-+         +-,-cx-+ +-,-cy-+ +-,-style--+ +-,-text-+
```

20.12. createWhiteFrame (UserDialog method)

The *createWhiteFrame* method creates a white *frame* static control in the dialog template. The frame is drawn with the current window background color, which is white in the default color scheme. This is the basic syntax:

```
>>--createWhiteFrame(--+------+--x-,-y-,-cx-,-cy--+-----------+-)--------------><
                       +--id--+                   +-,--style--+
```

20.13. createWhiteRect (UserDialog method)

The *createWhiteRect* method creates a white *rectangle* static control in the dialog template. The rectangle is filled with the current window background color, which is white in the default color scheme. The basic syntax is:

```
>>--createWhiteRect(--+------+--x-,-y-,-cx-,-cy--+-----------+-)---------------><
                      +--id--+                   +-,--style--+
```

20.14. connectStaticEvent (dialog object method)

To connect event notifications from an static control use the *connectStaticEvent*() method of the *dialog* object. The basic syntax is:

```
>>--connectStaticEvent(--id--,--event--+----------------+--)-------------------><
                                      +--,-methodName--+
```

20.15. newStatic (dialog object method)

Static objects can not be instantiated by the programmer from Rexx code. Rather an Static object is obtained by using the *newStatic* method of the *dialog* object. The syntax is:

```
>>--newStatic(--id--)-----------------------------><
```

20.16. getIcon

```
>>--getIcon---------------------------------------><
```

Gets the static control's icon image, if there is one. Only the image type of static control can have an icon.

Arguments:

This method takes no arguments.

Return value:

The current icon*Image* object for the control, if there is one. If the control does not have an icon set, then .nil is returned.

Details:

Programmers should manage the image objects as they think best. See the *Image* documentation for a discussion of this. The static control does not make a copy of the icon, nor does it release an icon.

Example:

This example determines if the icon image for the static control has been set. If not, the image is loaded and set.

```
staticControl = self~newStatic(IDC_ST_ICON)

icon = staticControl~getIcon
if icon == .nil then do
  size = .Size~new(48, 48)
  icon = .Image~getImage("appIcon", .Image~toID(IMAGE_ICON), size)
  staticControl~setIcon(icon)
end
```

20.17. getImage

```
>>--getImage(--+--------+--)---------------------><
               +--type--+
```

Returns the static control's image, if there is one. Only the image type static control will have an image.

Arguments:

The only argument is:

type [optional]

Specifies the type of the image: bitmap, icon, cursor, or enhanced metafile. You can use the *Image toID* to get the correct numeric value for one of the following symbols:

IMAGE_BITMAP	IMAGE_ICON
IMAGE_CURSOR	IMAGE_ENHMETAFILE

The default is IMAGE_BITMAP.

The programmer does not have to use .Image~toID() to get the numeric value for type. The correct number itself can be used. In general, symbolic IDs are used to make code more readable and less prone to error. However, since the value of IMAGE_CURSOR is 2, for example, the programmer could use 2 directly for the type argument.

Return value:

This method returns the *Image* object for the control, if there is one. If no image has been set, then .nil is returned.

Details:

Raises syntax errors when incorrect arguments are detected.

Programmers should manage the image objects as they think best. See the *Image* documentation for a discussion of this. The static control does not make a copy of the image, nor does it release an image.

Example:

This example gets the image from a static control and releases the image.

```
::method onExit
  expose staticPicture

  image = staticPicture~getImage(.Image~toID(IMAGE_CURSOR))
  if \ image~isNull then image~release
```

20.18. setIcon

```
>>--setIcon(--newImage--)------------------------><
```

Sets or removes the icon image for this static control. This method only effects the image type static control. For instance an icon can not be set for a frame or rectangle.

Arguments:

The single argument is:

newImage [required]

An*Image* object that represents the image for the control, or .nil to remove an existing image. The image object must be an icon image.

Return value:

This method returns the existing image object, if there is one. Otherwise .nil is returned.

Details:

Raises syntax errors when incorrect arguments are detected.

The programmer should manage the image objects as he thinks best. See the *Image* documentation for a discussion of this. The static control does not make a copy of the icon, nor does it release an icon image.

Example:

```
icon = .Image~getImage(.Image~toID(IDI_QUESTION), .Image~toID(IMAGE_ICON))
if \ icon~isNull then do
  oldIcon = iconControl~setIcon(icon)
  ...
end
else do
  -- handle error
  ...
end
```

20.19. setImage

```
>>--setImage(--imageObject--)--------------------><
```

Sets or removes the image for the static control. An image can only be set with the image type static control.

Arguments:

The arguments are:

imageObject [required]

An*Image* object that represents the image for the control, or .nil to remove an existing image. The image object can be any image type, including an icon image.

Return value:

This method returns the existing image object, if there is one. Otherwise .nil is returned.

Details:

Raises syntax errors when incorrect arguments are detected.

Programmers should manage the image objects as they think best. See the *Image* documentation for a discussion of this. The static control does not make a copy of the image, nor does it release an image.

Example:

```
image = .Image~getImage("Camera.bmp", .Image~toID(IMAGE_BITMAP))
if \ image~isNull then do
  oldImage = staticControl~setImage(image)
  if oldImage \== .nil then oldImage~release
  ...
end
else do
  -- handle error
  ...
end
```

StatusBar Controls

A status bar is a horizontal window at the bottom of a dialog that the programmer can use to display status information about the application. Status bars can be divided into more than one part to display different types of status information. Status bars set their initial size and position automatically, ignoring any coordinates the programmer may have specified. By default they position themselves at the bottom edge of the dialog with a width as wide as the dialog and a height that is calculated based on the metrics for the font of the dialog.

Unlike other dialog controls, many resource editors do not have built-in support for status bars. However it is easy to manually edit the .rc file and add a status bar. One technique is to add another control, say a combo box control, save the .rc file and then edit the combo box control definition in the .rc file to make it a status bar.

As an example of this, the following listing shows part of a .rc file with a combo box, that is intended to be edited to change it to a status bar;

```
LANGUAGE LANG_NEUTRAL, SUBLANG_NEUTRAL
IDD_DLL_TEST DIALOGEX 0, 0, 227, 114
STYLE DS_3DLOOK | DS_CENTER | DS_MODALFRAME | DS_SHELLFONT | WS_CAPTION | WS_VISIBLE |
 WS_POPUP | WS_SYSMENU
CAPTION "DLL Dialog"
FONT 8, "Ms Shell Dlg", 400, 0, 1
{
    CONTROL         "", IDC_CB, WC_COMBOBOX, WS_TABSTOP | NOT WS_VISIBLE | CBS_DROPDOWN |
 CBS_HASSTRINGS, 10, 33, 47, 12
    CONTROL         "", IDC_EDIT, WC_EDIT, NOT WS_BORDER | WS_TABSTOP | NOT WS_VISIBLE |
 ES_AUTOHSCROLL, 10, 55, 59, 12
    CONTROL         "Test", IDC_PB_TEST, WC_BUTTON, WS_TABSTOP | BS_PUSHBUTTON |
 BS_SPLITBUTTON, 10, 72, 70, 18
    CONTROL         "OK", IDOK, WC_BUTTON, WS_TABSTOP | BS_DEFPUSHBUTTON, 112, 72, 50, 14
    CONTROL         "Cancel", IDCANCEL, WC_BUTTON, WS_TABSTOP | BS_PUSHBUTTON, 167, 72, 50,
 14
}
```

Once the template is started as above, it is a simple matter to replace the IDC_CB line with the proper values for a status bar as below:

```
LANGUAGE LANG_NEUTRAL, SUBLANG_NEUTRAL
IDD_DLL_TEST DIALOGEX 0, 0, 227, 114
STYLE DS_3DLOOK | DS_CENTER | DS_MODALFRAME | DS_SHELLFONT | WS_CAPTION | WS_VISIBLE |
 WS_POPUP | WS_SYSMENU
CAPTION "DLL Dialog"
FONT 8, "Ms Shell Dlg", 400, 0, 1
{
    CONTROL         "", IDC_STATUSBAR, STATUSCLASSNAME, SBARS_TOOLTIPS, 0, 1, 227, 21
    CONTROL         "", IDC_EDIT, WC_EDIT, NOT WS_BORDER | WS_TABSTOP | NOT WS_VISIBLE |
 ES_AUTOHSCROLL, 10, 55, 59, 12
    CONTROL         "Test", IDC_PB_TEST, WC_BUTTON, WS_TABSTOP | BS_PUSHBUTTON |
 BS_SPLITBUTTON, 10, 72, 70, 18
    CONTROL         "OK", IDOK, WC_BUTTON, WS_TABSTOP | BS_DEFPUSHBUTTON, 112, 72, 50, 14
    CONTROL         "Cancel", IDCANCEL, WC_BUTTON, WS_TABSTOP | BS_PUSHBUTTON, 167, 72, 50,
 14
}
```

As noted, the status bar will ignore the size and position coordinate, so any 4 values will suffice. This technique will work for the ooDialog programmer wishing to use a status bar in their *RcDialog* or *ResDialog* dialogs.

The other good option for adding status bars to **`RcDialog`** or **`ResDialog`** dialogs is to use the *createStatusBarWindow* method of the *CreateWindows* class.

The **`StatusBar`** class provides methods to work with and manipulate the underlying Windows status bar dialog control which it represents. It is a concrete subclass of the dialog *control* object and therefore has all methods of the dialog control object.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with status bar controls:

Instantiation:

Use the *newStatusBar* method of the *dialog* object to retrieve a new StatusBar object.

Dynamic Definition:

To dynamically define a status bar in a *UserDialog* class, use the *createStatusBar* method.

Event Notification

To connect the *event* notifications sent by the underlying status bar control to a method in the Rexx dialog object use the *connectStatusBarEvent* method.

Window Creation:

To dynamically create a status bar window in a dialog, use the *createStatusBarWindow* method. This can only be done after the underlying Windows dialog has been created.

21.1. Method Table

The following table provides links to the documentation for the primary classes, methods, and attributes used in working with StatusBar objects, including the pertinent classes, and methods from other classes:

Table 21.1. Important StatusBar Methods

Method	Description
Useful	**External Methods**
newStatusBar	Returns a Rexx **`StatusBar`** object for the control with the specified ID.
createStatusBar	Creates a status bar control in the dialog template of a *UserDialog*.
createStatusBarWindow	Creates a status bar control in an existing Windows dialog.
connectStatusBarEvent	Connects status bar event notifications to a method in the Rexx dialog object
Instance Methods	
getBorders	Returns an array of three integers containing the borders of the status bar.
getIcon	Retrieves the icon image for the specified part.
getParts	Returns the number of parts in the status bar, and optionally, the coordinate of the right edge of each part.
getRect	Returns a *Rect* object that contains the bounding rectangle of the specified part in this status bar.
getText	Retrieves the text of the specified part and optionally some extra information related to the part.
getTextLength	Gets the length of the text for the specified part of the status bar, and optionally the drawing technique of the text.
getTipText	Retrieves the ToolTip text for the specified part in this status bar.
isSimple	Determines if this status bar is in *simple* mode.
setBkColor	Sets the background color in this status bar.

Method	Description
setIcon	Sets the icon for the specified part in this status bar.
setMinHeight	Sets the minimum height of a status bar's drawing area.
setParts	Sets the number of parts in a status window and the width of each part.
setText	Sets the text for the specified part.
setTipText	Sets the ToolTip text for a part in this status bar.
simple	Specifies whether a status window displays simple text or displays all window parts set by a previous *setParts* method.

21.2. newStatusBar (dialog object method)

StatusBar objects can not be instantiated by the programmer from Rexx code. Rather a StatusBar object is obtained by using the *newStatusBar*() method of the *dialog* object. The syntax is:

```
>>-newStatusBar(--id--)--------------------------><
```

21.3. createStatusBar (UserDialog method)

A status bar control can be added to the dialog template for a *UserDialog* dialog through the *createStatusBar*() method. The basic syntax is:

```
>>--createStatusBar(-id-,--x-,--y-,--cx-,--cy-+---------+--+-----------------+--)--><
                                              +-,-style-+  +-,-attributeName-+
```

21.4. connectStatusBarEvent (dialog object method)

To connect event notifications from a status bar control use the *connectStatusBarEvent*() method of the *dialog* object. The basic syntax is:

```
>>-connectStatusBarEvent(--id--,--event--+----------------+--+--------------+--)---><
                                         +--,-methodName--+  +-,-willReply--+
```

21.5. createStatusBarWindow (CreateWindows method)

A status bar control window can be created and added to any dialog through the *createStatusBarWindow*() method. The basic syntax is:

```
>>--createStatusBarWindow(--id--+----------+--)--><
                                +-,-style--+
```

21.6. getBorders

```
>>--getBorders-----------------------------------><
```

Returns an array of three integers containing the borders of the status bar. The meaning of the three integers is explained in the *Remarks* section.

Arguments:

There are no arguments to this method.

Return value:

Returns an array containing the borders on success, or the **`.nil`** object on error.

Remarks:

The item at index 1 is the width of the horizontal border, the item at index 2 is the width of the vertical border, and the item at index 3 is the width of the border between rectangles. The borders determine the spacing between the outside edge of the window and the rectangles within the window that contain text. The borders also determine the spacing between rectangles.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example ...

```
::method onTest2 unguarded

  sb = self~newStatusBar(IDC_STATUS)
  a = sb~getBorders
  say 'Width Horizontal Border:' a[1]
  say 'Width Vertical Border  :' a[2]
  say 'Spacing Border         :' a[3]
  say

/* Output might be:

Width Horizontal Border: 0
Width Vertical Border  : 2
Spacing Border         : 2

*/
```

21.7. getIcon

```
>>--getIcon(--index--)---------------------------><
```

Retrieves the icon image for the specified part.

Arguments:

The single argument is:

index [required]

The one-based index of the part whose icon is needed.

Return value:

Returns the icon for the specified part, or the **`.nil`** object if the part does not have an icon.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example comes from an application that assigns an icon to every part in the status bar. When the dialog is closed, each icon is retrieved and released to free up system resources:

```
::method leaving

  status = self~newStatusBar(IDC_STATUSBAR)
  do i = 1 to status~getParts
    icon = status~getIcon(i)
    icon~release
  end
```

21.8. getParts

```
>>--getParts(--+-----------+--)------------------><
               +--margins--+
```

Returns the number of parts in the status bar, and optionally, the coordinate of the right edge of each part.

Arguments:

The single argument is:

margins [optional] [in / out]

An **`Array`** object. If this argument is used, on return the indexes 1 through the number of parts will contain the right margin of that part. I.e., index 1 will contain the right margin of part 1, index 2 will contain the right margin for part 2, etc.. If any index is -1, the position of the right edge for that part extends to the right edge of the window.

Return value:

Returns the number of parts in the status bar.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example gets the number of parts in a status bar and also the right edge of each part:

```
::method onTest2 unguarded

  sb = self~newStatusBar(IDC_STATUS)

  margins = .array~new
  count = sb~getParts(margins)
  say 'Status bar has' count 'parts.'
  do i = 1 to count
    say '  part' i 'right edge at'   margins[i]
  end
  say

/* Output might be:

Status bar has 4 parts.
  part 1 right edge at 50
  part 2 right edge at 100
  part 3 right edge at 200
  part 4 right edge at -1
```

```
*/
```

21.9. getRect

```
>>--getRect(--index--)---------------------------><
```

Returns a *Rect* object that contains the bounding rectangle of the specified part in this status bar.

Arguments:

The single arguments is;

index [required]

The one-based index of the part whose bounding rectangle is needed.

Return value:

Returns the bounding rectangle on success, or the **.nil** object on error.

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example attempts to get the bounding rectangle for part 6 to part 2, descending, for a status bar. The status bar only has 4 parts:

```
::method onTest2 unguarded

  sb = self~newStatusBar(IDC_STATUS)

  do i = 6 to 2 by -1
    r = sb~getRect(i)
    say 'Rect for part' i':' r
  end
  say

/* Output might be:

Rect for part 6: The NIL object
Rect for part 5: The NIL object
Rect for part 4: a Rect (202, 2, 325, 23)
Rect for part 3: a Rect (102, 2, 200, 23)
Rect for part 2: a Rect (52, 2, 100, 23)

*/
```

21.10. getText

```
>>--getText(--index--+---------+--)--------------><
                     +-,-info--+
```

Retrieves the text of the specified part and optionally some extra information related to the part.

Arguments:

The arguments are:

index [required]

The one-based index of the part whose text is needed.

info [optional] [in / out]

A **`Directory`** object in which some extra information can be retrieved. If this argument is used, on return the object will contain these indexes:

DRAWTYPE:

A key word that specifies the type of operation used to draw the text. This will be exactly one of the following keywords:

LOWERBORDERS

The text is drawn with a border to appear lower than the plane of the window.

NOBORDERS

The text is drawn without borders.

NOTABPARSING

Tab characters are ignored. See the *Remarks* section.

OWNERDRAW

The text is drawn by the parent window. Owner draw is not currenly supported by the ooDialog framework.

POPOUT

The text is drawn with a border to appear higher than the plane of the window.

RTLREADING

The text will be displayed in the opposite direction to the text in the parent window.

TEXTLENGTH:

The length of the text string.

TEXT:

The text of the specified part. This will be the same string as the return value for this method.

Return value:

The text for the specified part.

Remarks

By default text is left-aligned. This can be changed by embedding tab characters in the text. Text to the right of a single tab character is center-aligned. Text to the right of 2 tab characters is right-aligned.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example retrieves the text and the extra info for all the parts in a status bar:

```
::method onTest2 unguarded
```

```
  sb = self~newStatusBar(IDC_STATUS)

  do i = 1 to sb~getParts
    info = .directory~new
    text = sb~getText(i, info)
    say 'Text for part' i':' text
    say '  Text length:   ' info~textLength
    say '  Text draw type:' info~drawType
    say '  Text (again):  ' info~text
    say
  end
  say

/* Output might be:

Text for part 1: Ok
  Text length:    2
  Text draw type: LOWERBORDERS
  Text (again):   Ok

Text for part 2: Test
  Text length:    4
  Text draw type: LOWERBORDERS
  Text (again):   Test

Text for part 3: Friday
  Text length:    6
  Text draw type: LOWERBORDERS
  Text (again):   Friday

Text for part 4: Viking Victory
  Text length:    14
  Text draw type: LOWERBORDERS
  Text (again):   Viking Victory

*/
```

21.11. getTextLength

```
>>--getTextLength(--index--+---------+--)--------><
                           +-,-info--+
```

Gets the length of the text for the specified part of the status bar, and optionally the drawing technique of the text.

Arguments:

The arguments are:

index [required]

The index of the part whose text length is to be retrieved.

info [optional] [in / out]

A **`Directory`** object in which the drawing technique can be retrieved. If this argument is used, on return the object will contain these indexes:

DRAWTYPE:

A key word that specifies the type of operation used to draw the text. This will be exactly one of the following keywords:

LOWERBORDERS
The text is drawn with a border to appear lower than the plane of the window.

NOBORDERS
The text is drawn without borders.

NOTABPARSING
Tab characters are ignored. See the *setText* method for an explanation of how embedded tabs are used in the text string.

OWNERDRAW
The text is drawn by the parent window. Owner draw is not currenly supported by the ooDialog framework.

POPOUT
The text is drawn with a border to appear higher than the plane of the window.

RTLREADING
The text will be displayed in the opposite direction to the text in the parent window.

TEXTLENGTH:
The length of the text string for the part.

Return value:
The length of the text for the specified part, or -1 on error.

Details
Raises syntax errors when incorrect usage is detected.

21.12. getTipText

```
>>--getTipText(--index--)------------------------><
```

Retrieves the ToolTip text for the specified part in this status bar.

Arguments:
The single argument is:

index [required]
The one-based index of the part whose ToolTip text is to be retrieved.

Return value:
Returns the tool tip text on success, or the empty string if there is no tool tip text.

Remarks:
Status bars must have the TOOLTIPS style to have tool tips.

Details
Raises syntax errors when incorrect usage is detected.

21.13. isSimple

```
>>--isSimple-------------------------------------><
```

Determines if this status bar is in *simple* mode.

Arguments:

This method has no arguments.

Return value:

Returns true if this status bar is in simple mode, false if it is not.

Remarks:

A status bar in simple mode displays only one part. In addition, when the text of the status bar is set, the operating system use a technique to redraw the text that reduces flicker. The *simple* method is used to switch a status bar between modes.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows a code snippet that is used in an application that has a basic mode and an advanced mode. The code snippet is used to toggle between the 2 modes:

```
...
sb = self~newStatusBar(IDC_STATUSBAR)

if sb~isSimple then do
    self~statusBarUpdates = 'off'
    self~setSimpleTasks
    self~useWizrds = .true
end
else do
    self~statusBarUpdates = 'on'
    self~useAdvancedTasks
    self~useWizrds = .false
end
```

21.14. setBkColor

```
>>--setBkColor(--colorRef--)---------------------><
```

Sets the background color in this status bar.

Arguments:

The single argument is:

colorRef [required]

A *COLORREF* value that specifies the new background color for the status bar. Use the *colorRef* class method of the *Image* class to construct a proper COLORREF. Use CLR_DEFAULT to set the background to the status bar's default color.

Return value:

Returns the previous background color, or CLR_DEFAULT if the status bar was using its default color.

Remarks:

Many dialog controls have a set background color API. However, when themes are turned on, it usually appears that the set background color has no effect. Such is the case with the status bar control. Under most Windows 7 themes, the *setBkColor* method appears to have no effect. That the method works can be seen by changing the current theme to the Windows Classic them.

Details

Raises syntax errors when incorrect usage is detected.

21.15. setIcon

```
>>--setIcon(--icon--,--index--)------------------><
```

Sets the icon for the specified part in this status bar.

Arguments:

The arguments are:

icon [required]

The icon *Image* for the part, or the **`.nil`** object to remove the current icon for the part.

index [required]

The one-based index of the part whose icon is being changed.

Return value:

Returns the old icon, or the **`.nil`** object if there was no existing icon, on success. Returns -1 on error.

Remarks:

Some dialog controls take ownership of image resources assigned to them and take care of releasing the resources. This is not the case for the status bar. The ooDialog programmer is responsible for releasing the icon images when they are no longer needed.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows a status bar with 4 parts. An image list is used with 4 icons in it. After loading the image list and setting up the status bar, the first 4 icons in the image list are assigned to the 4 parts in the status bar:

```
::method initDialog
    expose imageList status
    ...
    self~loadImageList
    status = self~newStatusBar(IDC_STATUSBAR)
    ...

    do i = 1 to 4
        icon = imageList~getIcon(i)
        status~setIcon(icon, i)
    end
```

21.16. setMinHeight

```
>>--setMinHeight(--minHeight--)------------------><
```

Sets the minimum height of a status bar's drawing area.

Arguments:

The single argument is:

minHeight

The minimum height, in pixels, that is desired.

Return value:

Returns 0, always.

Remarks:

MSDN says the minimum height is set to: *minHeight* plus twice the width of the vertical border. The *getBorders* method can be used to get the width of the vertical border.

However, experimentation seems to show that the minimum height is actually set to: *minHeight* plus 1 * the width of the vertical border. But MSDN is vague as to what exactly is getting set. The doc says: *minimum height of a status window's drawing area.* The drawing area of a window usually means the client area of a window. Testing shows that the client area height is set to *minHeight* plus the width of the vertical border.

Details

Raises syntax errors when incorrect usage is detected.

Example:

In this example, the minimum height of the client area of the status bar is set to 28 pixels. The actual height was verified using Spy++

```
sb = self~newStatusBar(IDC_STATUS)

-- We want the minimum height to be 28 pixels.
min = 28
borders = sb~getBorders

min -= borders[2]

sb~setMinHeight(min)
```

21.17. setParts

```
>>--setParts(--rightEdges--)---------------------><
```

Sets the number of parts in a status window and the width of each part.

Arguments:

The single argument is:

rightEdges [required]

An array of positive integers. Each integer specifies the right edge of a part in client *coordinates*, in pixels. The array can not be sparse. The number of items in the array will be the number parts created. A -1 specifies the remaining area of the status bar.

Return value:

True on success, false on error.

Remarks:

It is important to note that the integers in the area represent the coordinate of the right edge of the part, not the width of the part. A mistake the author often makes. For example, if a status bar has a width of 300, to divide it into 3 equal parts, use an array of 100, 200, -1. Not an array of 100, 100, -1.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example divides the status bar up into 4 parts. The first 3 parts are each 75 pixels in width and the fourth part takes of the remainder of the status bar. Its width will be dependent on the width of the status bar. If the width of the status bar was only 200 pixels, the fourth part would not show at all and the third part would only be 50 pixels wide:

```
    status = self~newStatusBar(IDC_STATUSBAR)

    parts = .array~of(75, 150, 225, -1)
    status~setParts(parts)
```

21.18. setText

```
>>--setText(--text--,--index--+------------+--)---------------><
                              +-,-drawType--+
```

Sets the text for the specified part.

Arguments:

The arguments are:

text [required]

The text for the part.

index [required]

The one-based index of the part whose text is to be set.

drawType [optional]

A single keyword that specifies the drawing technique used by the control to draw the text. The following keywords are accepted, case is not significant:

LOWERBORDERS

The text is drawn with a border to appear lower than the plane of the window. This is the default if the *drawType* argument is omitted.

NOBORDERS

The text is drawn without borders.

NOTABPARSING
Tab characters are ignored. See the *Remarks* section.

OWNERDRAW
The text is drawn by the parent window. Owner draw is not currenly well supported by the ooDialog framework.

POPOUT
The text is drawn with a border to appear higher than the plane of the window.

RTLREADING
The text will be displayed in the opposite direction to the text in the parent window.

Return value:
Returns true on success, false on error.

Remarks:
By default text is left-aligned. This can be changed by embedding tab characters in the text. Text to the right of a single tab character is center-aligned. Text to the right of 2 tab characters is right-aligned.

Details
Raises syntax errors when incorrect usage is detected.

Example:
This example divides a status bar into 4 parts and then sets the text for the parts

```
status = self~newStatusBar(IDC_STATUSBAR)

parts = .array~of(75, 150, 225, -1)
status~setParts(parts)
status~setText("Friday", 1, 'POPOUT')
status~setText("January", 2, 'POPOUT')
status~setText("22nd", 3)
status~setText("Viking Victory", 4)
```

21.19. setTipText

```
>>--setTipText(--text--,--index--)---------------><
```

Sets the ToolTip text for a part in this status bar.

Arguments:
The arguments are:

text [required]
The text of the tool tip.

index [required]
The one-base index of the part whose tool tip is being set.

Return value:
Returns 0, always.

Remarks:

The length of *text* can not be longer than 511. The status bar must have the TOOLTIPS style to display a tool tip. The tool tip is only displayed under these circumstances:

- When the corresponding part in the status bar contains only an icon.
- When the corresponding pane in the status bar contains text that is truncated due to the size of the pane.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example divides a status bar up into 4 parts. Since the first 3 parts are not wide enough to display the entire text string, a tool tip is added for each of them:

```
    status = self~newStatusBar(IDC_STATUSBAR)

    parts = .array~of(75, 150, 225, -1)
    status~setParts(parts)

    text = "Thank God it is Friday"
    status~setText(text, 1, 'POPOUT')
    status~setTipText(text, 1)

    text = "January in northern Canada"
    status~setText(text, 2, 'POPOUT')
    status~setTipText(text, 2)

    text = "The 22nd Bomber Squadron"
    status~setText(text, 3)
    status~setTipText(text, 3)

    status~setText("Viking Victory", 4)
```

21.20. simple

```
>>--simple(--+---------+--)----------------------><
             +--onOff--+
```

Specifies whether a status window displays simple text or displays all window parts set by a previous *setParts* method.

Arguments:

The single argument is:

onOff [optional]

True to put the status bar in simple mode, false to take it out of simple mode. The default if the argument is omitted is true.

Return value:

Returns 0, always.

Remarks:

A status bar in simple mode displays only one part. In addition, when the text of the status bar is set, the operating system use a technique to redraw the text that reduces flicker. The *isSimple* method can be used to test what mode the status bar is currently in. In addition the *SIMPLEMODECHANGE* event can be connected for the status bar. The event handler will be invoked each time the status bar mode switches to or from simple.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This simple code snippet merely toggles the simple mode each time it is executed:

```
if sb~isSimple then do
  sb~simple(.false)
end
else do
  sb~simple(.true)
end
```

Tab Controls

A tab in a tab control is similar to a label in a file cabinet or divider in a notebook. Using a tab control the programmer can define several pages for the same area in a dialog. When the user selects one of the tabs in the tab control, the page for that tab is displayed. Individual pages can contain their own set of dialog controls or information.

A tab control consists of the tabs and a display area. The tabs occupy a row (or rows) along the edge of the display area. Each tab consists of a label and an icon. Tabs can be added and removed. Tabs are identified by their index. In ooDialog, the indexes are zero-based rather than one-based as is normal in Rexx. Because of backwards compatibility, this can not be changed.

There is a special type of tab control with tabs that look like buttons. Clicking a button immediately performs a command instead of displaying a page.

As usual in Windows controls, there are a number of different styles for the tab control. For instance, by default a tab control will only display one row of tabs. However, there is the MULTILINE style. A tab control with this style will use multiple rows to display the tabs, so that all tabs are visible at once.

The **Tab** class provides methods to work with and manipulate the underlying Windows tab dialog control which it represents. It is a concrete subclass of the dialog *control* object and therefore has all methods of the dialog control object.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with tab controls:

Instantiation:

Use the *newTab* method of the *dialog* object to retrieve a new Tab object.

Dynamic Definition:

To dynamically define a tab in a *UserDialog* class, use the *createTab* method.

Event Notification

To connect the *event* notifications sent by the underlying tab control to a method in the Rexx dialog object use the *connectTabEvent* method.

22.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with Tab objects, including the pertinent methods from other classes:

Table 22.1. Important Tab Methods

Method	Description
Useful	**External Methods**
newTab	Returns a **Tab** object for the control with the specified ID.
createTab	Creates a tab control in the dialog template of a *UserDialog*
connectTabEvent	Connects tab event notifications to a method in the Rexx dialog object
Instance Methods	
addFullSeq	Inserts a sequence of 1 through N tabs into a tab control. The index for an icon in the imagelist for the tab can be included.
addSequence	Inserts a sequence of 1 through N tabs into a tab control. The index for an icon can not be included.

Method	Description
adjustToRectangle	Calculates the window rectangle of a tab control that corresponds to the specified display rectangle.
calcDisplayRect	Takes the window rectangle of the tab control, and adjusts the rectangle to that needed for the display rectangle
calcWindowRect	Takes a display rectangle and adjusts the rectangle to be that needed for tab control to contain the display rectangle.
delete	Removes a tab from a tab control.
deleteAll	Removes all tabs from a tab control.
focus	Sets the focus to the specified tab in a tab control.
focused	Returns the index of the tab that has the focus.
getImageList	Retrieves the current image list from the tab control, if there is one.
getItemRect	Gets the bounding rectangle for a tab in a tab control.
insert	Inserts a new tab in a tab control.
itemInfo	Retrieves information about a tab in a tab control.
items	Retrieves the number of tabs in a tab control.
last	Retrieves the index of the last tab in a tab control.
modify	Sets some, or all, of the attributes of a tab.
posRectangle	Retrieves the rectangle around a tab in a tab control.
requiredWindowSize	Calculates the display rectangle of a tab control that corresponds to the specified window rectangle.
rows	Retrieves the current number of rows of tabs in a tab control.
select	Selects the tab with the specified label text.
selected	Retrieves the label text of the currently selected tab.
selectedIndex	Retrieves the index of the currently selected tab.
selectIndex	Selects a tab in the tab control.
setImageList	Assigns, or removes, an image list for the tab control.
setItemSize	Sets the width and height of the tabs in a fixed-width tab control.
setPadding	Sets the amount of space (padding) around the icon and the label of a tab.
setSize	Sets the width and height of the tabs in a fixed-width tab control.

22.2. newTab (dialog object method)

Tab objects can not be instantiated by the programmer from Rexx code. Rather a Tab object is obtained by using the *newTab*() method of the *dialog* object. The syntax is:

```
>>-newTab(--id--)---------------------><
```

22.3. createTab (UserDialog method)

A tab control can be added to the dialog template for a *UserDialog* dialog through the *createTab*() method. The basic syntax is:

```
>>--createTab(-id-,--x-,--y-,--cx-,--cy-+---------+--+----------------+--)--->< 
                                        +-,-style-+  +-,-attributeName-+
```

22.4. connectTabEvent (dialog object method)

To connect event notifications from an tab control use the *connectTabEvent*() method of the *dialog* object. The basic syntax is:

```
>>-connectTabEvent(--id--,--event--+---------------+--+-------------+--)-----><
                                   +--,-methodName--+  +-,-willReply--+
```

22.5. addFullSeq

```
>>--addFullSeq(--text1--,-+-------+-,-+-------+-+-------------------------+--)--><
                          +-icon1-+   +-user1-+ +-,-...-+-----------+-+----+
                                                        +--,-textN--+ +-..-+
```

The *addFullSeq* method inserts a sequence of 1 through N tabs into a tab control. For each inserted tab, an optional icon index and user object can be specified.

Arguments:

The arguments are:

text [required]

The label text for the inserted tab.

icon1 [optional]

The index of the icon in the image list of the tab control, set with the *setImageList*() method.

user1 [optional]

Any object the programmer wants stored together with the tab. This can be used to save information to be associated with each tab.

Return value:

The index of the tab inserted last, or -1 for any error.

Remarks:

The argument *text* can repeat any number of times. A single comma can used to signal that the icond index and / or user data argument is omitted for a tab. For each text argument one tab is inserted into the tab control. However, in the sequence 1 though N arguments, the text argument can not be omitted. Another way of saying that is, if two commas follow the text argument, the third argument is required. This is acceptable:

```
tabControl~addSequence(myText, , , myText2, ,)
```

But this is not, and will raise a syntax error:

```
tabControl~addSequence("Label One", , , "Label Two", , , )
```

If no arguments are used, -1 is returned.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

The following example adds a sequence of tabs and sets their label and icon index, but omits any user data for each tab:

```
::method initDialog

  tab = self~newTab("IDC_TAB")
  if tab \= .nil then do
    tab~setImageList(imageList)
    tab~addFullSeq("s11", 0, , "s12", 1, , "s13", 2, , "s14", 3)
  end
```

22.6. addSequence

```
>>--addSequence(--text1--+--------------------+--)---------------------------><
                         +--,-...--+----------+
                                   +--,-textN--+
```

The *addSequence* method inserts a sequence of one through N tabs in a tab control. The sequence of tabs starts immediately after the last added tab, or at index 0 if there are no tabs in the tab control.

Arguments:

The only, repeating, argument is:

text

The label text for the inserted tab(s).

Return value:

The index of the tab inserted last, or -1 for any error.

Remarks:

The argument *text* can repeat any number of times. For each argument one tab is inserted into the tab control. However, in the sequence 1 though N arguments, no argument can be omitted. If no arguments are used, -1 is returned.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

The following example inserts three tabs in a tab control:

```
::method initDialog

  tab = self~newTab("IDC_TAB")
  if tab \= .nil then do
    tab~addSequence("First Tab", "Second Tab", "Third Tab")
  end
```

22.7. adjustToRectangle

```
>>--adjustToRectangle(--left--,--top--,--right--,--bottom--)-><
```

The adjustToRectangle method calculates the window rectangle of a tab control that corresponds to the specified display rectangle.

Arguments:

The arguments are:

left

The x-coordinate of the upper left corner of the display rectangle.

top

The y-coordinate of the upper left corner of the display rectangle.

right

The x-coordinate of the lower right corner of the display rectangle.

bottom

The y-coordinate of the lower right corner of the display rectangle.

Return value:

A string containing the coordinates of the window rectangle, or an empty string. The coordinates are separated by blanks and are in the following order:

- X-coordinate of the upper left corner of the rectangle
- Y-coordinate of the upper left corner of the rectangle
- X-coordinate of the lower right corner of the rectangle
- Y-coordinate of the lower right corner of the rectangle

Remarks:

This method is outmoded and retained solely for backwards compatibility. All new code should use the *calcWindowRect* method.

22.8. calcDisplayRect

```
>>calcDisplayRect(--rect--)--------------------><
```

Given a *Rect* object that specifies the window rectangle of a tab control, calculates the display area of the tab control. This method should be used when the size of the tab control is fixed and the size of the contents of a page in the tab control can be adjusted.

Arguments:

The single argument is:

rect [required] [in/out]

A *Rect* object that specifies the window rect of the tab control when the method is invoked. On return, the **`rect`** object will contain the coordinates of the display area of the tab control.

Return value:

Returns 0, always.

Remarks:

Two strategies can be used for placing the content of a page of a tab control. The position and size of the content can be adjusted to the position and size of the display area of the tab control, or the tab control's position and size can be adjusted so the content fits into the display area. This method is used to facilitate the first strategy. Give the size and position of the tab control, it returns the position and size for the page content.

This method is only for tab controls that have their tabs on the top. It does not apply for tab controls that have their tabs on the sides or bottom.

Details

Raises syntax errors when incorrect arguments are detected.

22.9. calcWindowRect

```
>>--calcWindowRect(--rect--)---------------------><
```

Takes a display rectangle and adjusts the rectangle to be the window rect of the tab control needed for that display.

Arguments:

The single argument is:

rect [required] [in/out]

A *Rect* object that specifies the display area of the tab control when the method is invoked. On return, the **`rect`** object will contain the coordinates of the window rectangle of the tab control.

Return value:

This method always returns 0.

Remarks:

Two strategies can be used for placing the content of a page of a tab control. The position and size of the content can be adjusted to the position and size of the display area of the tab control, or the tab control's position and size can be adjusted so the content fits into the display area. This method is used to facilitate the second strategy. Give the position and size of the page content, it returns the position and size that the tab control needs to be.

This method is only for tab controls that have their tabs on the top. It does not apply for tab controls that have their tabs on the sides or bottom.

Details

Raises syntax errors when incorrect arguments are detected.

22.10. delete

```
>>--delete(--index--)----------------------------><
```

The delete method removes a tab from a tab control.

Arguments:

The only argument is:

tab

The number of the tab to be removed.

Return value:

0
: The tab was removed.

-1
: You did not specify *tab* or there is no tab available.

1
: For all other cases.

22.11. deleteAll

```
>>--deleteAll------------------------------------><
```

The deleteAll method removes all tabs from a tab control.

Return value:

0
: The tabs were removed.

1
: For all other cases.

22.12. focus

```
>>--focus(--index--)-----------------------------><
```

The focus method sets the focus to the specified tab in a tab control.

Arguments:

The only argument is:

index
: The index of the tab to receive the focus.

Return value:

0.

22.13. focused

```
>>--focused--------------------------------------><
```

The focused method returns the number of the tab that has the focus. The tab with the focus can differ from the selected tab.

Return value:

The number of the tab having the focus.

22.14. getImageList

```
>>--getImageList---------------------------------><
```

Retrieves the current image list from the tab control, if there is one.

Arguments:

There are no arguments to this method.

Return value:

This method returns the current image *ImageList*, if there is one. **`.nil`** is returned if there is no current image list.

Example:

This example gets the current image list from the tab control when the dialog is closed and releases it as it is no longer userd.

```
::method ok
  self~releaseImageLists
  return self~ok:super

::method cancel
  self~releaseImageLists
  return self~cancel:super

::method releaseImageLists
  expose tabControl

  imageList = tabControl~getImageList
  if imageList \== .nil then imageList~release
```

22.15. getItemRect

```
>>--getItemRect(--index--,--rect--)--------------><
```

Gets the bounding rectangle for a tab in a tab control.

Arguments:

The arguments are:

index [required]

The index of the tab to get the bounding rectangle for.

rect [required] [in/out]

A *Rect* object in which the coordinates of tab are returned.

Return value:

Returns true on success, otherwise false.

22.16. insert

```
>>--insert(--+-----+--+--------+--+--------+--+------------+--)---------------><
```

```
               +-tab-+  +-,-text-+  +-,-icon-+  +-,-userData-+
```

Inserts a new tab in a tab control.

Arguments:

The arguments are:

tab [optional]

The index of the tab to insert. If you omit this argument, the index of the last tab added to the control, plus one, is used for the index. If the argument is omitted, and no tab has yet been added to the tab control, the index of zero is ued.

text [optiona]

The label text for the inserted tab. If omitted, the empty string is used.

icon [optional]

The index of the icon in the image list of the tab control to use with the tab. The image list for the control is set with the *setImageList*() method. If this argument is omitted, no icon is used for the tab.

userData [optiona]

Any object the programmer wants to store with the tab.

Return value:

The index of the insert tab, or -1 for any error.

Example:

The following example inserts three tabs in a tab control with the specified label and index in the image list for an icon for the tab:

```
::method initDialog

  tab = self~newTab("IDC_TAB")
  if tab \= .Nil then do
    tab~setImageList(imageList)
    tab~insert(, "First Tab", 0)
    tab~insert(, "Second Tab", 1)
    tab~insert(, "Third Tab", 2)
  end
```

22.17. itemInfo

```
>>--itemInfo(--index--)--------------------------><
```

The itemInfo method retrieves information about a tab in a tab control.

Arguments:

The only argument is:

text

The number of the tab.

Return value:

A compound variable that stores the attributes of the tab, or -1 in all other cases. The compound variable can be:

RetStem.!TEXT
: The label text for the tab.

RetStem.!IMAGE
: The index of the tab in the image list of the tab control, or -1 if the tab does not have an image.

RetStem.!PARAM
: An integer value stored together with the tab to save information:

Example:

The following example displays the text of all tabs:

```
::method DisplayText
  curTab = self~newTab("ID_TAB")
  if curTab \= .Nil then do
    do i = 0 to curTab~items - 1
       ItemInfo. = curTab~itemInfo(i)
       say ItemInfo.!Text
    end
  end
```

22.18. items

```
>>--items----------------------------------------><
```

The items method retrieves the number of tabs in a tab control.

Return value:

The number of the tabs, or 0 for all other cases.

Example:

The following example displays the number of tabs:

```
::method DisplayTabNum
  curTab = self~newTab("ID_TAB")
  if curTab \= .Nil then do
    say curTab~items
  end
```

22.19. last

```
>>--last-----------------------------------------><
```

Retrieves the index of the last tab in a tab control.

Return value:

The index of the last tab. -1 is returned if there are no tabs or for any error.

22.20. modify

```
>>--modify(--index--+--------+--+--------+--+-----------+--)----------------><
                    +-,-text-+  +-,-icon-+  +-,-userData-+
```

The *modify* method sets some, or all, of the attributes of a tab. It can be used to change the value of any attribute that has already been set.

Arguments:

The arguments are:

index [required]

The index of the tab to modify.

text [optiona]

The new label text for the inserted tab.

icon [optional]

The new index of the icon in the image list of the tab control to use with the tab. The image list for the control is set with the *setImageList*() method.

userData [optiona]

The new object the programmer wants to store with the tab. There is no restriction on this argument, it can be any object.

Return value:

0 for success and 1 for any error, or if all the optional arguments are omitted.

22.21. posRectangle

```
>>--posRectangle(--index--)----------------------><
```

The posRectangle method retrieves the rectangle around a tab in a tab control.

Arguments:

The only argument is:

index

The index of the tab.

Return value:

A string containing the coordinates of the rectangle, or an empty string. The coordinates are separated by blanks and are in the following order:

- X-coordinate of the upper left corner of the rectangle
- Y-coordinate of the upper left corner of the rectangle
- X-coordinate of the lower right corner of the rectangle
- Y-coordinate of the lower right corner of the rectangle

22.22. requiredWindowSize

```
>>--requiredWindowSize(--left--,--top--,--right--,--bottom--)-><
```

The requiredWindowSize method calculates the display rectangle of a tab control that corresponds to the specified window rectangle.

Arguments:

The arguments are:

left

The x-coordinate of the upper left corner of the window rectangle.

top

The y-coordinate of the upper left corner of the window rectangle.

right

The x-coordinate of the lower right corner of the window rectangle.

bottom

The y-coordinate of the lower right corner of the window rectangle.

Return value:

A string containing the coordinates of the display rectangle, or an empty string. The coordinates are separated by blanks and are in the following order:

- X-coordinate of the upper left corner of the rectangle
- Y-coordinate of the upper left corner of the rectangle
- X-coordinate of the lower right corner of the rectangle
- Y-coordinate of the lower right corner of the rectangle

Remarks:

This method is outmoded and retained solely for backwards compatibility. All new code should use the *calcDisplayRect* method.

22.23. rows

```
>>--rows-----------------------------------------><
```

The rows method retrieves the current number of rows of tabs in a tab control. Only tab controls with multiline style can have several rows of tabs.

Return value:

The number of the tab rows.

22.24. select

```
>>--select(--text--)-----------------------------><
```

The select method selects the tab with the specified label text.

Arguments:

The only argument is:

text

The label text of the tab to be selected.

Return value:

The number of the selected tab, or 0 in all other cases.

22.25. selected

```
>>--selected-------------------------------------><
```

The selected method retrieves the label text of the currently selected tab.

Return value:

The label text of the currently selected tab, or 0 in all other cases.

22.26. selectedIndex

```
>>--selectedIndex--------------------------------><
```

The selectedIndex method retrieves the number of the currently selected tab.

Return value:

The number of the currently selected tab, or 0 in all other cases.

22.27. selectIndex

```
>>--selectIndex(--index--)-----------------------><
```

Selects a tab in the tab control.

Arguments:

The only argument is:

index [required]

The index of the tab to be selected.

Return value:

The index of the previously selected tab on success, or -1 for any error.

Remarks:

The tab control does not send a SELCHANGING or SELCHANGE *event* notification when a tab is selected using this method.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example allows the user to push a button to go to the next or previous tabs. Note that the *onNewTab* method is connected to the SELCHANGE event. Since the tab control does not generate the SELCHANGE event when the *selectIndex* method is invoked, the program just calls the *onNewTab* method directly.

```
::method defineDialog
```

```
  self~createTab(IDC_TAB, 10, 5, 305, 265)

  self~createPushButton(IDC_PB_PREVIOUS, 10, 278, 60, 14, , "Previous Control",
 onPrevious)
  self~createPushButton(IDC_PB_NEXT, 80, 278, 60, 14, , "Next Control", onNext)

  self~connectTabEvent(IDC_TAB, SELCHANGE, onNewTab)

  ...

::method onNext
  expose tabControl

  tabControl~selectIndex(tabControl~selectedIndex + 1)
  self~onNewTab


::method onPrevious
  expose tabControl

  tabControl~selectIndex(tabControl~selectedIndex - 1)
  self~onNewTab
```

22.28. setImageList

```
>>--setImageList(--newImageList--)---------------><
```

Assigns, or removes, an image list for the tab control. Using **`.nil`** for the first argument removes the current image list. Each tab can have an icon associated with it, which is specified by an index in the image list for the tab control.

Destroying a tab control does not destroy an image list that is associated with it. The programmer must destroy the image list separately, if desired. This is useful if the programmer wants to assign the same image list to multiple tab controls. In essence, the ownership of the image list remains with the programmer. The *ImageList* and *Image* classes are used to manage image lists and images in ooDialog. The documentation on both classes discusses when and why the programmer may want to release image lists. The Image class documentation has the most detail on this subject.

Arguments:

The single argument is

newImageList [required]

The image *ImageList* to assign to the tab control. If this argument is **`.nil`** the existing image list, if any, is removed.

Return value:

The existing image list is returned, if there is one. Otherwise, **`.nil`** is returned.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example creates an image list from a bitmap file that is a series of 16x16 bitmaps, each one a colored letter. The image list is then assigned to the tab control.

```
  -- Add all the tabs, including the index into the image list for an icon for
```

```
   -- each tab.
   tc~addFullSeq("Red", 0, ,"Green", 1, , "Moss", 2, , "Blue", 3, , "Purple", 4, , -
                 "Cyan", 5, , "Gray", 6)

   -- Create a COLORREF (pure white) and load our bitmap.  The bitmap is a
   -- series of 16x16 images, each one a colored letter.
   cRef = .Image~colorRef(255, 255, 255)
   image = .Image~getImage("bmp\psdemoTab.bmp")

   -- Create our image list, as a masked image list.
   flags = .DlgUtil~or(.Image~toID(ILC_COLOR24), .Image~toID(ILC_MASK))
   imageList = .ImageList~create(.Size~new(16, 16), flags, 10, 0)

   if \image~isNull,  \imageList~isNull then do
      imageList~addMasked(image, cRef)
      tc~setImageList(imageList)

      -- The image list makes a copy of each image added to it.  So, we can now
      -- release the original image to free up some small amount of system
      -- resources.
      image~release
   end
   else do
      -- do some type of error handling
   end
```

22.29. setItemSize

```
Form 1:

>>--setItemSize(--sizeObj--)---------------------><


Form 2:

>>--setItemSize(--cx--,--cy--)-------------------><


Generic form:

>>--setItemSize(--newSize--)---------------------><
```

Sets the width and height of the tabs in a fixed-width tab control.

Arguments:

The arguments are:

newSize [required]

The new size (height and width) in pixels. The *newSize* argument(s) can be specified either as a single *Size* object, or as two separate arguments, width first and height second.

Return value:

The return is a **`Size`** object specifying the old width and height of the tabs in the control.

Remarks:

The *setSize* and *setItemSize* methods are identical except that the *setSize* method returns the old size a string and the *setItemSize* method returns the old size as a *Size* object.

Details

Raises syntax errors when incorrect arguments are detected.

22.30. setPadding

```
Form 1:

>>--setPadding(--sizeObj--)----------------------><


Form 2:

>>--setPadding(--cx--,--cy--)--------------------><


Generic form:

>>--setPadding(--padding--)----------------------><
```

The *setPadding* method sets the amount of space (padding) around the icon and the label of a tab in the tab control.

Arguments:

The arguments are:

padding [required]

The new padding (horizontal and vertical padding) in pixels. The *padding* argument(s) can be specified either as a single *Size* object, or as two separate arguments, horizontal padding first and vertical padding second.

Return value:

Returns 0, always.

22.31. setSize

```
Form 1:

>>--setSize(--sizeObj--)-------------------------><


Form 2:

>>--setSize(--cx--,--cy--)-----------------------><


Generic form:

>>--setSize(--newSize--)-------------------------><
```

The *setSize* method sets the width and height of tabs in a fixed-width tab control.

Arguments:

The arguments are:

newSize [required]

The new size (height and width) in pixels. The *newSize* argument(s) can be specified either as a single *Size* object, or as two separate arguments, width first and height second.

Return value:

The old width and height, in pixels, as a string of two blank-delimited words. The width is the first word and the height is the second word.

Remarks:

The *setItemSize* and *setSize* methods are identical except that the *setSize* method returns the old size as a string and the *setItemSize* method returns the old size as a *Size* object.

Details

Raises syntax errors when incorrect arguments are detected.

ToolBar Controls

xxx

yyy

The **`ToolBar`** class provides methods to work with and manipulate the underlying Windows toolbar dialog control which it represents. It is a concrete subclass of the dialog *control* object and therefore has all methods of the dialog control object.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with toolbar controls:

Instantiation:

Use the *newToolBar* method of the *dialog* object to retrieve a new ToolBar object.

Dynamic Definition:

To dynamically define a toolbar in a *UserDialog* class, use the *createToolBar* method.

Event Notification

To connect the *event* notifications sent by the underlying toolbar control to a method in the Rexx dialog object use the *connectToolBarEvent* method.

23.1. Method Table

The following table provides links to the documentation for the primary classes, methods, and attributes used in working with ToolBar objects, including the pertinent classes, and methods from other classes:

Table 23.1. Important Tab Methods

Method	Description
Useful	**Classes**
TbButton	**`TbButton`** objects define the buttons in a toolbar and are used to receive information concerning buttons in the toolbar.
Useful	**External Methods**
newToolBar	Returns a **`ToolBar`** object for the control with the specified ID.
createToolBar	Creates a toolbar control in the dialog template of a *UserDialog*
connectToolBarEvent	Connects toolbar event notifications to a method in the Rexx dialog object
Instance Methods	
addBitmap	x.

23.2. newToolBar (dialog object method)

ToolBar objects can not be instantiated by the programmer from Rexx code. Rather a ToolBar object is obtained by using the *newToolBar*() method of the *dialog* object. The syntax is:

```
>>-newToolBar(--id--)----------------------------><
```

23.3. createToolBar (UserDialog method)

A toolbar control can be added to the dialog template for a *UserDialog* dialog through the *createToolBar*() method. The basic syntax is:

```
>>--createToolBar(-id-,--x-,--y-,--cx-,--cy-+---------+--+----------------+--)--->< 
                                            +-,-style-+  +-,-attributeName-+
```

23.4. connectToolBarEvent (dialog object method)

To connect event notifications from an toolbar control use the *connectToolBarEvent*() method of the *dialog* object. The basic syntax is:

```
>>-connectToolBarEvent(--id--,--event--+---------------+--+-------------+--)----><
                                       +--,-methodName--+  +-,-willReply--+
```

23.5. addBitmap

```
>>--addBitmap(--+--------+--)--------------------------------------------><
                +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

23.6. addButtons

```
>>--addButtons(--+--------+--)--------------------------------------------><
```

```
                  +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

23.7. addString

```
>>--addString(--+--------+--)--------------------------------------------><
                  +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

23.8. autoSize

```
>>--autoSize(--+--------+--)-------------------------------------------><
               +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.9. buttonCount

```
>>--buttonCount(--+--------+--)----------------------------------------><
                  +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.10. changeBitmap

```
>>--changeBitmap(--+--------+--)---------------------------------------------><
                   +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.11. checkButton

```
>>--checkButton(--+--------+--)---------------------------------------------><
                  +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.12. commandToIndex

```
>>--commandToIndex(--+--------+--)---------------------------------------------><
                     +--type--+
```

xx

Arguments:

The arguments are:

TERM
    xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.13. customize

```
>>--customize(--+--------+--)---------------------------------------------><
                     +--type--+
```

xx

Arguments:

The arguments are:

TERM
    xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.14. getButton

```
>>--getButton(--+--------+--)--------------------------------------------><
                +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.15. getButtonText

```
>>--getButtonText(--+--------+--)--------------------------------------------><
                    +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

23.16. getButtonTextEx

```
>>--getButtonTextEx(--+--------+--)--------------------------------------------><
                      +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

23.17. getDisabledImageList

```
>>--getDisabledImageList(--+--------+--)-------------------------------------------><
                           +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.18. getExtendedStyle

```
>>--getExtendedStyle(--+--------+--)-------------------------------------------><
                       +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.19. getHotImageList

```
>>--getHotImageList(--+--------+--)--------------------------------------------><
                      +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.20. getImageList

```
>>--getImageList(--+--------+--)--------------------------------------------><
                   +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

23.21. getPressedImageList

```
>>--getPressedImageList(--+--------+--)--------------------------------------------><
                          +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

23.22. indexTocommand

```
>>--indexTocommand(--+--------+--)--------------------------------------------><
                     +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.23. insertButton

```
>>--insertButton(--+--------+--)---------------------------------------------><
                   +--type--+
```

xx

Arguments:

The arguments are:

TERM
    xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.24. isButtonChecked

```
>>--isButtonChecked(--+--------+--)---------------------------------------------><
                   +--type--+
```

xx

Arguments:

The arguments are:

TERM
    xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.25. isButtonEnabled

```
>>--isButtonEnabled(--+--------+--)---------------------------------------------><
                      +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.26. isButtonHidden

```
>>--isButtonHidden(--+--------+--)---------------------------------------------><
                      +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.27. isButtonHighlighted

```
>>--isButtonHighlighted(--+--------+--)-------------------------------------------><
                          +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.28. isButtonIndeterminate

```
>>--isButtonIndeterminate(--+--------+--)---------------------------------------------><
                            +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.29. isButtonPressed

```
>>--isButtonPressed(--+--------+--)---------------------------------------------><
                      +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.30. loadImages

```
>>--loadImages(--+--------+--)---------------------------------------------><
                 +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.31. markButton

```
>>--markButton(--+--------+--)---------------------------------------------><
                 +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.32. setBitmapSize

```
>>--setBitmapSize(--+--------+--)---------------------------------------------><
                    +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

23.33. setButtonText

```
>>--setButtonText(--+--------+--)---------------------------------------------><
                    +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

23.34. setDisabledImageList

```
>>--setDisabledImageList(--+--------+--)--------------------------------------------><
                           +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

23.35. setExtendedStyle

```
>>--setExtendedStyle(--+--------+--)--------------------------------------------><
                       +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:
xx

Remarks:
Additional comments.

Details
Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:
This example ...

```

```

23.36. setHotImageList

```
>>--setHotImageList(--+--------+--)--------------------------------------------><
                      +--type--+
```

xx

Arguments:
The arguments are:

TERM
xx

Return value:
xx

Remarks:
Additional comments.

Details
Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:
This example ...

```

```

23.37. setImageList

```
>>--setImageList(--+--------+--)--------------------------------------------><
                      +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

23.38. setPressedImageList

```
>>--setPressedImageList(--+--------+--)-------------------------------------------><
                          +--type--+
```

xx

Arguments:

The arguments are:

TERM
: xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

23.39. TbButton Class

A *TbButton* object represents a button in a *ToolBar*. The attributes of a *TbButton* object contain information for a specific button in a toolbar.

xx

23.39.1. Method Table

The following table lists the class and instance methods of the **`TbButton`** class:

Table 23.2. TbButton Class Method Reference

Method	Description
Class Methods	**Class Methods**
new	Instantiates a new **`TbButton`** object.
new	*new*
Attribute Methods	**Attribute Methods**
bitmapID	xx.
cmdID	xx.
itemData	xx.
state	xx.
style	xx.
text	xx.
Instance Methods	**Instance Methods**
assignBitmapID	xx.

23.39.2. new (Class Method)

```
>>--new(--+--------+--)-------------------------------------------><
          +--type--+
```

xx

Arguments:

xx

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details:

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```

```

23.39.3. bitmapID (Attribute)

```
>>--bitmapID-----------------------------------------------------><

>>--bitmapID = varName-------------------------------------------><
```

xx

bitmapID get:

details about get

bitmapID set:

details about set

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example ...

```

```

23.39.4. cmdID (Attribute)

```
>>--cmdID--------------------------------------------------------><

>>--cmdID = varName----------------------------------------------><
```

xx

cmdID get:

details about get

cmdID set:

details about set

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example ...

```

```

23.39.5. itemData (Attribute)

```
>>--itemData-----------------------------------------------------><

>>--itemData = varName-------------------------------------------><
```

xx

itemData get:
details about get

itemData set:
details about set

Remarks:
Additional comments.

Details
Raises syntax errors when incorrect usage is detected.

Example:
This example ...

```
```

23.39.6. state (Attribute)

```
>>--state--------------------------------------------------------><

>>--state = varName----------------------------------------------><
```

xx

state get:
details about get

state set:
details about set

Remarks:
Additional comments.

Details
Raises syntax errors when incorrect usage is detected.

Example:
This example ...

```
```

23.39.7. style (Attribute)

```
>>--style-----------------------------------------------------><

>>--style = varName-------------------------------------------><
```

xx

style get:
details about get

style set:
details about set

Remarks:
Additional comments.

Details
Raises syntax errors when incorrect usage is detected.

Example:
This example ...

```
```

23.39.8. text (Attribute)

```
>>--text------------------------------------------------------><

>>--text = varName--------------------------------------------><
```

xx

text get:
details about get

text set:
details about set

Remarks:
Additional comments.

Details
Raises syntax errors when incorrect usage is detected.

Example:
This example ...

```
```

23.39.9. assignBitmapID

```
>>--assignBitmapID(--+--------+--)---------------------------------------------><
                     +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example ...

```
```

ToolTip Controls

Tooltip controls are pop-up windows that display text. Typically the text describes a *tool*. A *tool* is either a window or an application defined area within a window. Conceptually, a ToolTip is a dialog control that contains tools. Each ToolTip can contain any number of *tools*. A ToolTip with no added tool will never display text. In ooDialog, the tools added to a ToolTip are usually other dialog controls.

ToolTips are hidden most of the time. They appear automatically, or pop up, when the user pauses the mouse pointer over a tool. The ToolTip appears near the pointer and disappears when the user clicks a mouse button or moves the pointer away from the tool.

ToolTip controls can display single or multiple lines of text. They can have square or rounded corners. They might or might not have a stem that points to the tools like a cartoon balloon. ToolTip text can be stationary or can move with the mouse pointer, these are called tracking ToolTips. Stationary text can be displayed adjacent to a tool or it can be displayed over a tool, which is referred to as in-place ToolTips. Standard ToolTips are stationary, display a single line of text, have square corners, and have no stem pointing to the tool.

The **`ToolTip`** class provides methods to work with and manipulate the underlying Windows ToolTip dialog control which it represents. It is a concrete subclass of the dialog control *object* and therefore has all methods of the dialog control object.

The following classes, and methods from other classes, are needed, or are useful, when working with ToolTip controls:

ToolInfo Class

The *ToolInfo* class represents the information a ToolTip control uses to manage the *tools* it contains. The attributes of a **`ToolInfo`** object represent the information specific to a single tool within a ToolTip. Many of the methods of the **`ToolTip`** class require a **`ToolInfo`** object as an argument, or return information using a **`ToolInfo`** object.

Creation:

Unlike most other types of dialog controls, a ToolTip can not be added to a dialog *template*. ToolTips are created dynamically using the *createToolTip* method. ToolTips can only be created after the underlying Windows dialog exists. Therefore, ToolTips can not be created in the *defineDialog* method of the *UserDialog* class.

Instantiation:

Use the *newToolTip* method of the *dialog* object to retrieve a Rexx ToolTip object.

Event Notification

To connect the *event* notifications sent by the underlying ToolTip control to a method in the Rexx dialog object use the *connectToolTipEvent* method.

24.1. Tool Identification

Each ToolTip can contain more than one *tool*. Because of this, the ToolTip control needs a way of uniquely identifying each individual tool. In the operating system, each tool is identified using a combination of a window *handle* and an ID. The ID however, can be either another window handle or a unique number. When the ID is a number, the number is similar to a *resource ID*.

For the operating system, the first part of the 2-part identifier is the handle of the window that the ToolTip will send notifications to. The second part of the identifier can be a second window handle, if the ToolTip is going to subclass that window, or an unique number if the ToolTip is not going to subclass the tool. This may seem somewhat complicated to the Rexx programmer, especially if the programmer has little experience with the underlying details of programming dialog controls.

The ooDialog framework attempts to simplify this somewhat by allowing the Rexx programmer to identify an unique tool in several different ways. For the most common use of ToolTips, identifying a tool is simple and can be done with one Rexx object. However, ooDialog also supplies the means needed to identify a tool when more advanced use of ToolTips is desired.

This documentation refers to a tool identifier as a *Rexx object combination*. This can be a combination of two Rexx objects, but quite often only one object is needed to identify a tool. **`ToolTip`** methods that require a tool to be identified, have a required argument, a *toolHwnd* argument, followed by a second optional argument a *toolID* argument. The *toolHwnd* argument is usually argument 1, followed by the *toolID* argument. However, in some cases the *toolHwnd* argument may be argument 2 followed by the *toolID* as argument 3.

The combination of the two arguments, *toolHwnd* and *toolID*, is always handled in the same way, which is as follows:

if *hwndTool* == a **`ToolInfo`** object

A *ToolInfo* object always uniquely identifies a tool. When the *toolHwnd* argument is a **`ToolInfo`** object that is sufficient and the second *toolID* argument is ignored.

if *hwndTool* \== a **`ToolInfo`** object and *toolID* is omitted

It is very common that a ToolTip tool is a dialog control. For example quite often ToolTips are used for push buttons, so that some label can be displayed when the user hovers the mouser over the button. ooDialog makes it easy for the Rexx programmer to identify this type of tool by using just the dialog control object. When the *toolHwnd* argument is not a **`ToolInfo`** object, and the *toolID* is omitted, then the *toolHwnd* argument must be a dialog control object. This dialog control object is the tool identifier. The ooDialog framework deduces the proper values to identify the tool to the operating system.

if *hwndTool* \== a **`ToolInfo`** object and *toolID* is not omitted

This combination of identifying objects is needed when some more advanced uses of ToolTips are desired. It allows the programmer to specify the exact values that should be passed on to the operarting system to identify the tool. For this combination, *hwndTool* can be either a dialog object, or a dialog control object. When *hwndTool* is a dialog object, then *toolID* must be either a dialog control object or an unique whole number. The first part of the 2-part operating system ID for the tool is set to the window handle of the Rexx object, the dialog window handle or the dialog control window handle.

The *toolID* argument can be either a dialog control, or an unique non-negative whole number. If the *toolHwnd* argument is a dialog, then *toolID* can be a dialog control or a number. However, if *toolHwnd* is a dialog control, the *toolID* must be a number. The second part of the 2-part operating system identifier is set to the window handle of the dialog control, if *toolID* is a dialog control. Otherwise the second part is set to the number specified. Note that numbers can be specified using *symbolic* IDs just as other types of resource IDs are.

24.2. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with ToolTip objects, including the pertinent methods from other classes:

Table 24.1. Important ToolTip Methods

Method	Description
Useful	**Classes**

Method	Description
ToolInfo	A `ToolInfo` object is used to supply tool information to, or receive tool information from, the ToolTip control.
Useful	**External Methods**
createToolTip	Creates the underlying Windows ToolTip control and returns an instantiated Rexx ToolTip object.
connectToolTipEvent	Connects tool tip event notifications to a method in the Rexx dialog object
newToolTip	Returns the Rexx `ToolTip` object for the tool tip control with the specified ID.
Instance Methods	**Instance Methods**
activate	Activates or deactivates this ToolTip.
addTool	Adds a tool to this ToolTip.
addToolEx	Adds the tool, specified by a `ToolInfo` object, to this ToolTip.
addToolRect	Adds a tool that uses a rectangular area in the dialog as its trigger point.
adjustRect	Calculates a ToolTip control's text display rectangle from its window rectangle, or the ToolTip window rectangle needed to display a specified text display rectangle.
delTool	Removes the specified tool from this ToolTip.
enumTools	Retrieves a *ToolInfo* object for the tool at the specified index.
getBubbleSize	Returns the width and height, as a *Size* object, of this ToolTip control.
getCurrentToolInfo	Retrieves a *ToolInfo* object whose attributes represent the current tool in this tool tip.
getDelayTime	Retrieves one of the 3 delay times currently set for this ToolTip.
getMargin	Returns a rectangle that describes the margins of the ToolTip.
getMaxTipWidth	Retrieves the maximum width for this ToolTip window.
getText	Retrieves the text information this ToolTip control maintains about the specified tool.
getTipBkColor	Retrieves the background color for this ToolTip's window.
getTipTextColor	Retrieves the text color fot this ToolTip's window.
getTitle	Retrieves information concerning the title and icon of this ToolTip control.
getToolCount	Retrieves the number of tools this ToolTip contains.
getToolInfo	Retrieves the information, as a *ToolInfo* object, that this tool tip control maintains about the specified tool.
hasCurrentTool	Tests if this ToolTip has a current tool.
hitTestInfo	Tests a point to determine whether it is within the bounding rectangle of a tool within the window specified and, if it is, retrieves information about the tool.
manageAtypicalTool	Initiates the management of a ToolTip tool that is a dialog control.
newToolRect	Sets a new bounding rectangle for a tool.
pop	Removes, hides, this ToolTip's display window.
popup	Causes this ToolTip to display at the coordinates of the last mouse message.

Method	Description
setDelayTime	Sets the initial, pop-up, and reshow durations for this ToolTip control.
setMargin	Sets the top, left, bottom, and right margins for this ToolTip window. A margin is the distance, in pixels, between the ToolTip window border and the text contained within the ToolTip window.
setMaxTipWidth	Sets the maximum width, in pixels, for this ToolTip window.
setTipBkColor	Sets the background color for this ToolTip window.
setTipTextColor	Sets the text color for this ToolTip window.
setTitle	Adds a title string and optionally an icon image to this ToolTip.
setToolInfo	Sets the information that this ToolTip control maintains for a tool.
setWindowTheme	Sets the visual style of a ToolTip control.
trackActivate	Activates or deactivates a tracking ToolTip.
trackPosition	Sets the position of a tracking ToolTip.
update	Forces the current tool to be redrawn.
updateTipText	Sets the ToolTip text for a tool.

24.3. connectToolTipEvent (dialog object method)

To connect event notifications from an tool tip control use the *connectToolTipEvent*() method of the *dialog* object. The basic syntax is:

```
>>--connectToolTipEvent(--id--,--event--+----------+--+------------+--)------><
                                        +-,-mthName-+  +-,-willReply-+
```

24.4. createToolTip (dialog object method)

To create the underlying Windows ToolTip control use the *createToolTip* method of the *dialog* object. This method also returns the instantiated Rexx object that represents the created tool tip control. The basic syntax is:

```
>>--createToolTip(--id--+---------+--)------------><
                        +-,-style--+
```

24.5. newToolTip (dialog object method)

ToolTip objects can not be instantiated by the programmer from Rexx code using the normal *new* method. Rather a ToolTip object is obtained either by using the *createToolTip* method or the *newToolTip* method of the dialog *object*. The syntax for the *newToolTip* is:

```
>>-newToolTip(--id--)--------------------><
```

24.6. activate

```
>>--activate(--+-----------+--)----------------><
```

```
                 +--activate--+
```

Activates or deactivates this ToolTip.

Arguments:

The single optional argument is:

activate [optional]

If true, activates this tool it, if false, deactivate this tool tip. The default if the argument is omitted is true.

Return value:

Returns 0, always.

Remarks:

A ToolTip control can be either active or inactive. When it is active, the ToolTip text appears when the mouse pointer is on a tool. When it is inactive, the ToolTip text does not appear, even if the pointer is on a tool.

Details

Raises syntax errors when incorrect usage is detected.

24.7. addTool

```
>>--addTool(--tool--+---------+--+---------+--+-----------+--)----------------><
                    +-,-text--+  +-,-flags-+  +-,-userData-+
```

Adds a tool to this ToolTip.

Arguments:

The arguments are:

tool [required]

The dialog control that defines the tool being added.

text [optional]

Text for the tool. If omitted, or the empty string, or the string: TEXTCALLBACK, then the tool tip sends the *NEEDTEXT* notification and the program supplies the text.

The length of the text must be less than 1024 characters, which includes any possible end of line (0x0D0A) sequences.

flags [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. If this argument is omitted, the flags are set to IDISHWND SUBCLASS:

ABSOLUTE	PARSELINKS	TRACK
CENTERTIP	RTLREADING	TRANSPARENT
IDISHWND	SUBCLASS	

ABSOLUTE

Positions the ToolTip window at the exact same coordinates specified by the *trackPosition* method. Without this flag the ToolTip control chooses where to display the ToolTip window based on the coordinates specified, which places the ToolTip close to the tool. This flag must be used with the TRACK flag.

CENTERTIP
: Centers the ToolTip window below the tool specified by the *tool* argument.

IDISHWND
: Indicates that the ID part of the tool *identification* is the window handle to the tool. If this flag is not set, the ID part is the tool's identification number. For the *addTool* method, the ooDialog framework always sets this flag, it does not need to be specified by the programmer. The window *handle* of the *tool* argument is always the ID part of the tool identification.

PARSELINKS
: Requires Common Control *Library* version 6.0 or later.

  Indicates that links in the ToolTip text should be parsed.

RTLREADING
: Indicates that the ToolTip text will be displayed in the opposite direction to the text in the parent window.

SUBCLASS
: Indicates that the ToolTip control should subclass the tool's window to intercept messages, such as WM_MOUSEMOVE. If this flag is not set, the *manageAtypicalTool* method must be used so the mouse messages are forwarded to the ToolTip control. For the simple case that the *addTool* method is designed for, the *manageAtypicalTool* method will not work. The ooDialog framwork always sets this flag during the *addTool* method, the programmer does not need to include this flag.

TRACK
: Positions the ToolTip window next to the tool to which it corresponds and moves the window according to coordinates supplied by the *trackPosition* method. The programmer must activate this type of tool using the *trackActivate* method.

TRANSPARENT
: Causes the ToolTip control to forward mouse event messages to the parent window. This is limited to mouse events that occur within the bounds of the ToolTip window.

userData [optional]
: A user data value to be associated with the tool. This can be any Rexx object the programmer wants. Note that the value is associated with the tool, not the tool tip.

Return value:

Returns true on success, false on error.

Remarks:

The *addTool* method is intended to be a convenient method for use in the most common case of adding a tool. The case where the tool is a simple dialog control. Use the *addToolRect* or the *addToolEx* methods to add a tool with characteristics that need to be more explicitly defined.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example creates a ToolTip and then adds a tool to it. The tool is defined using a push button dialog control and the text for the tool is specified at the time the tool is added:

```
pbTest = self~newPushButton(IDC_PB_TEST)
count  = 0
...

ttTest = self~createToolTip(IDC_TT_TEST)
ttTest~addTool(pbTest, "Push the Test button to start the regression testing.")
```

24.8. addToolEx

```
>>--addToolEx(--tool--)--------------------------><
```

Adds the specified tool to this ToolTip.

Arguments:

The single argument is:

tool [required]

A *ToolInfo* object that defines the tool being added.

Return value:

Returns true on success, false on error.

Remarks:

There are a number of different attributes that can be set when adding a tool to a tool tip. The *addToolEx* method is designed to let the programmer specify any valid combination of attributes allowed by the operating system. To do this, it requires the specifier of the tool to be a **ToolInfo** object. The programmer is responsible for setting the tool attriubtes as he wishes.

For the simple case of adding a tool that is a dialog control, the *addTool* is a more convenient method.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example creates a ToolTip and then adds a tool that will supply customized tool tips for a tree-view:

```
tv = self~newTreeView(IDC_TREE)
rect = tv~clientRect

tt = self~createToolTip(IDC_TT)

toolInfo = .ToolInfo~new(tv, ID_TREE_TOOL, '', 'TRANSPARENT', rect)
tt~addToolEx(toolInfo)
```

24.9. addToolRect

```
>>--addToolRect(--dlg-,-id-,-rect--+--------+-+---------+-+------------+--)----><
                                   +-,-txt--+ +-,-flags-+ +-,-userData-+
```

Adds a tool that uses a rectangular area in the dialog as its trigger point.

Arguments:

The arguments are:

dlg [required]

The dialog *object* that contains the rectangular area specified by the *rect* argument.

id [required]

An unique, non-negative, whole number that identifies the tool being added. May be numeric or *symbolic*.

rect [required]

A *Rect* object that specifies the bounding rectangle coordinates of the tool. The coordinates are in client *coordinates* of the dialog identified by the *dlg* argument.

txt [optional]

Text for the tool. If omitted, or the empty string, or the string: TEXTCALLBACK, then the ToolTip sends the *NEEDTEXT* notification and the program supplies the text.

The length of the text must be less than 1024 characters, which includes any possible end of line (0x0D0A) sequences.

flags [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. If this argument is omitted, the flags are set to SUBCLASS:

ABSOLUTE	PARSELINKS	TRACK
CENTERTIP	RTLREADING	TRANSPARENT
IDISHWND	SUBCLASS	

ABSOLUTE

Positions the ToolTip window at the exact same coordinates specified by the *trackPosition* method. Without this flag the ToolTip control chooses where to display the ToolTip window based on the coordinates specified, which places the ToolTip close to the tool. This flag must be used with the TRACK flag.

CENTERTIP

Centers the ToolTip window below the tool specified by the *tool* argument.

IDISHWND

Indicates that the ID part of the tool *identification* is the window handle to the tool. This flag can not be used for the *addToolRect* method. If it is present, the ooDialog framework removes it.

PARSELINKS

Requires Common Control *Library* version 6.0 or later.

Indicates that links in the ToolTip text should be parsed.

RTLREADING

Indicates that the ToolTip text will be displayed in the opposite direction to the text in the parent window.

SUBCLASS

Indicates that the ToolTip control should subclass the tool's window to intercept messages, such as WM_MOUSEMOVE. If this flag is not set, some means must be used so that the mouse messages are forwarded to the ToolTip control. For the use case that the *addToolRect* method is designed for, this flag should always be set. The ooDialog framwork always sets this flag during the *addToolRect* method, the programmer does not need to include this flag.

TRACK

Positions the ToolTip window next to the tool to which it corresponds and moves the window according to coordinates supplied by the *trackPosition* method. The programmer must activate this type of tool using the *trackActivate* method.

TRANSPARENT

Causes the ToolTip control to forward mouse event messages to the parent window. This is limited to mouse events that occur within the bounds of the ToolTip window.

userData [optional]

A user data value to be associated with the tool. This can be any Rexx object the programmer wants. Note that the value is associated with the tool, not the tool tip.

Return value:

Returns true on success, false on error.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example was created solely as a demonstration of how ToolTips work, it has no real practical value. It displays ToolTips as the mouse moves over a dialog:

```
tt = self~createToolTip(IDC_TT_MAIN)

clRect = self~clientRect
hMidpoint = trunc((clRect~right - clRect~left) / 2) + clRect~left
vMidpoint = trunc((clRect~bottom - clRect~top) / 2) + clRect~top

clRect1 = .Rect~new(clRect~left, clRect~top, hMidpoint, vMidpoint)
clRect2 = .Rect~new(hMidpoint + 1, clRect~top, clRect~right, vMidpoint + 1)
clRect3 = .Rect~new(clRect~left, vMidpoint + 1, hMidpoint + 1, clRect~bottom)
clRect4 = .Rect~new(hMidpoint + 1, vMidpoint + 1, clRect~right, clRect~bottom)

text1 = 'Over main dialog, top left quadrant'
text2 = 'Over main dialog, top right quadrant'
text3 = 'Over main dialog, bottom left quadrant'
text4 = 'Over main dialog, bottom right quadrant'

ret = tt~addToolRect(self, IDTOOL_DLG_RECT1, clRect1, text1, 'TRANSPARENT')
ret = tt~addToolRect(self, IDTOOL_DLG_RECT2, clRect2, text2, 'TRANSPARENT')
ret = tt~addToolRect(self, IDTOOL_DLG_RECT3, clRect3, text3, 'TRANSPARENT')
ret = tt~addToolRect(self, IDTOOL_DLG_RECT4, clRect4, text4, 'TRANSPARENT')

self~connectToolTipEvent(IDC_TT_MAIN, 'SHOW', onShow)
```

24.10. adjustRect

```
>>--adjustRect(--rect--+-----------+--)----------><
                       +-,-larger--+
```

Calculates a ToolTip control's text display rectangle from its window rectangle, or the ToolTip window rectangle needed to display a specified text display rectangle.

Arguments:

The arguments are:

rect [required]

A *Rect* object used to specify the rectangle to adjust. On a successful return, the coordinates in the rectangle will be adjusted as specified by the *larger* argument.

larger [optional]

True or false to specify how the rectangle is adjusted. If omitted, the default is false.

If true, *rect* is used to specify a text-display rectangle and it is updated to contain the corresponding window rectangle. On return the rectangle is *larger* in this case. If false, *rect* is used to specify a window rectangle and it is updated to the corresponding text display rectangle. On return, the rectangle is *not* larger, (it is smaller,) in this case.

Return value:

Returns true if the rectangle was adjusted correctly, and returns false if an error occurred.

Remarks:

This method is particularly useful when a ToolTip control is used to display the full text of a string that is truncated. It is commonly used with list-view and tree-view controls. Typically send this method would be used in response to a *SHOW* SHOW event notification so that the ToolTip control properly can be positioned properly.

The ToolTip's window rectangle is somewhat larger than the text display rectangle that bounds the ToolTip string. The window origin is also offset up and to the left from the origin of the text display rectangle. To position the text display rectangle, the programmer must calculate the corresponding window rectangle and use that rectangle to position the ToolTip. The *adjustRect* method handles this calculation for the programmer.

If *larger* is set to true, *adjustRect* takes the size and position of the desired ToolTip text display rectangle, and passes back the size and position of the ToolTip window needed to display the text in the specified position. If *larger* is set to false, the programmer can specify a ToolTip window rectangle and the *adjustRect* method will return the size and position of its text rectangle.

Details

Raises syntax errors when incorrect usage is detected.

24.11. delTool

```
>>--delTool(--toolHwnd--+----------+--)----------><
                        +-,-toolID-+
```

Removes the specified tool from this ToolTip.

Arguments:

The arguments are:

toolHwnd [required]

The *toolHwnd* and *toolID* arguments are the Rexx object combination that *identify* the tool to be deleted.

toolID [optional]

The *toolHwnd* and *toolID* arguments are the Rexx object combination that *identify* the tool to be deleted.

Return value:

Returns 0, always.

Remarks:

The *Tool Identification* section explains exactly how the Rexx object combination is used to specify the tool to be deleted.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows a tool being added to a ToolTip and then at some later point in the application being deleted.

```
pbTest = self~newPushButton(IDC_PB_TEST)
count  = 0
...

ttTest = self~createToolTip(IDC_TT_TEST)
ttTest~addTool(pbTest, "Push the Test button to start the regression testing.")

...

ttTest~delTool(pbTest)
```

24.12. enumTools

```
>>--enumTools(--+---------+--)-------------------><
                +--index--+
```

Retrieves a *ToolInfo* object for the tool at the specified index.

Arguments:

The single argument is:

index

The one-based index of the tool to retrieve. If omitted, *index* defaults to 1.

Return value:

Returns the tool, as a **`ToolInfo`** object, at the index specified, or the **`.nil`** object if there is no tool at the index.

Remarks:

The Microsoft documentation says nothing about what, if any, order the tools contained in a ToolTip are in.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example *enumerates* the tools in a ToolTip:

```
do i = 1 to toolTip~getToolCount
  toolInfo = toolTip~enumTools(i)
  say 'Tool info  hwnd:     ' toolInfo~rexxHwnd
  say 'Tool info  id:       ' toolInfo~rexxID
  say 'Tool info  text:     ' toolInfo~text
  say 'Tool info  flags:    ' toolInfo~flags
  say 'Tool info  rect:     ' toolInfo~rect
  say 'Tool info  userData:' toolInfo~userData
  say 'Tool info  resource:' toolInfo~resource
  say
end
```

24.13. getBubbleSize

```
>>--getBubbleSize(--toolHwnd--+----------+--)----><
                              +-,-toolID-+
```

Returns the width and height, as a *Size* object, of this ToolTip control.

Arguments:

The arguments are:

toolHwnd [required]

The *toolHwnd* and *toolID* arguments are the Rexx object combination that *identify* the tool to be deleted.

toolID [optional]

The *toolHwnd* and *toolID* arguments are the Rexx object combination that *identify* the tool to be deleted.

Return value:

A **`Size`** object specifying the width and height of this tool tip control.

Remarks:

The *Tool Identification* section explains exactly how the Rexx object combination is used to specify the tool to be deleted.

Details

Raises syntax errors when incorrect usage is detected.

24.14. getCurrentToolInfo

```
>>--getCurrentToolInfo---------------------------><
```

Retrieves a *ToolInfo* object whose attributes represent the current tool in this tool tip.

Arguments:

There are no arguments for this method.

Return value:

Returns the current tool as a *ToolInfo* object, if one exists, or the `.nil` object if there is not a current tool.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

24.15. getDelayTime

```
>>--getDelayTime(--+---------+--)----------------><
                   +--which--+
```

Retrieves one of the 3 delay times currently set for this ToolTip.

Arguments:

The single argument is:

which [optional]

Exactly one of the following keywords, case is not significant. This specifies which of the 3 possible delay times to retrieve. The default if omitted is AUTOPOP:

AUTOPOP	INITIAL	RESHOW

AUTOPOP

The amount of time the ToolTip window remains visible if the pointer is stationary within a tool's bounding rectangle.

INITIAL

The amount of time the pointer must remain stationary within a tool's bounding rectangle before the ToolTip window appears.

RESHOW

The amount of time it takes for subsequent ToolTip windows to appear as the pointer moves from one tool to another.

Return value:

The specified delay time in milliseconds.

Details

Raises syntax errors when incorrect usage is detected.

24.16. getMargin

```
>>--getMargin------------------------------------><
```

Returns a rectangle that describes the margins of the ToolTip

Arguments:

This method has no arguments.

Return value:

A *Rect* object whose attributes specify the margins of the ToolTip.

Remarks:

The attributes of the returned rectangle do not define a bounding rectangle. For the purpose of this method, the attributes are interpreted as follows:

top:

Distance between top border and top of ToolTip text, in pixels.

left:

Distance between left border and left end of ToolTip text, in pixels.

bottom:

Distance between bottom border and bottom of ToolTip text, in pixels.

right:

Distance between right border and right end of ToolTip text, in pixels.

24.17. getMaxTipWidth

```
>>--getMaxTipWidth-------------------------------><
```

Retrieves the maximum width for this ToolTip window.

Arguments:

This method has no arguments.

Return value:

Returns a whole number value that represents the maximum ToolTip width, in pixels. If no maximum width was set previously, this method returns -1.

Remarks:

The maximum ToolTip width value does not indicate a ToolTip window's actual width. Rather, if a ToolTip string exceeds the maximum width, the control breaks the text into multiple lines, using spaces or newline characters to determine line breaks. If the text cannot be segmented into multiple lines, it will be displayed on a single line. The length of this line may exceed the maximum ToolTip width.

24.18. getText

```
>>--getText(--toolHwnd--+----------+--)----------><
                        +-,-toolID-+
```

Retrieves the text information this ToolTip control maintains about the specified tool.

Arguments:

The arguments are:

toolHwnd [required]
: The *toolHwnd* and *toolID* arguments are the Rexx object combination that *identify* the tool to be deleted.

toolID [optional]
: The *toolHwnd* and *toolID* arguments are the Rexx object combination that *identify* the tool to be deleted.

Return value:
The text string for the specified tool

Remarks:
The *Tool Identification* section explains exactly how the Rexx object combination is used to specify the tool to get the text for.

Details
Raises syntax errors when incorrect usage is detected.

Example:
This example example prints to the screen the text for a tool tip where the tool is a rectangular area in a dialog:

```
      say 'Text for the tool tip:' tt~getText(self, IDTOOL_DLG_RECT1)

/* Output might be:

Text for the tool tip: Over main dialog, top left quadrant

*/
```

24.19. getTipBkColor

```
>>--getTipBkColor--------------------------------><
```

Retrieves the background color for this ToolTip window.

Arguments:
There are no arguments for this method.

Return value:
Returns a *COLORREF* value that represents the background color of this ToolTip window.

24.20. getTipTextColor

```
>>--getTipTextColor------------------------------><
```

Retrieves the text color fot this ToolTip window.

Arguments:
This method does not have any arguments.

Return value:

Returns a *COLORREF* value that represents the background color of this ToolTip window.

24.21. getTitle

```
>>--getTitle-------------------------------------><
```

Retrieves information concerning the title and icon of this ToolTip control.

Arguments:

This method takes no arguments.

Return value:

Returns a **`.Directory`** object whose indexes contain the title and icon information for this ToolTip. Indexes are:

TITLE

The title text.

ICON

The ICON index will be the *Image* object used for the icon of this ToolTip, or one of the following keywords if a system image is used for the icon:

NONE	INFO	WARNINGLARGE
ERROR	INFOLARGE	
ERRORLARGE	WARNING	

NONE

No icon.

ERROR

Error icon.

ERRORLARGE

Large warning icon.

INFO

Info icon.

INFOLARGE

Large info icon.

WARNING

Warning icon.

WARNINGLARGE

Large warning icon.

ISKEYWORD

True if the ICON index is a keyword, false if it is an **`Image`** object.

Remarks:

The underlying Windows API for *getTitle* appears to be rather idiosyncratic.

In testing, when setting the icon to an actual icon image, getting the icon always returns the INFOLARGE keyword, rather than the icon image. In addition, when setting the icon to any of the LARGE keyword values, getting the icon always returns the non-large keyword. This anomaly is mentioned in several places on the web. The following table shows this behaviour:

Table 24.2. getTitle Anomalies

Actual Icon Value	getTitle() returns
hIcon	INFOLARGE
INFOLARGE	INFO
WARNINGLARGE	WARNING
ERRORLARGE	ERROR

24.22. getToolCount

```
>>--getToolCount---------------------------------><
```

Retrieves the number of tools this ToolTip contains.

Arguments:

There are no arguments for this method.

Return value:

Returns the number of tools in the ToolTip.

Remarks:

The *getToolCount* method in conjunction with the *enumTools* method is useful in enumerating the tools of the ToolTip.

Example:

This example prints out information for each tool in the ToolTip:

```
do i = 1 to tt~getToolCount
  toolInfo = tt~enumTools(i)
  say 'Tool info  hwnd:     ' toolInfo~rexxHwnd
  say 'Tool info  id:       ' toolInfo~rexxID
  say 'Tool info  text:     ' toolInfo~text
  say 'Tool info  flags:    ' toolInfo~flags
  say 'Tool info  rect:     ' toolInfo~rect
  say 'Tool info  userData:' toolInfo~userData
  say 'Tool info  resource:' toolInfo~resource
  say
end
```

24.23. getToolInfo

```
>>--getToolInfo(--toolHwnd--+----------+--)------><
                            +-,-toolID-+
```

Retrieves the information, as a *ToolInfo* object, that this tool tip control maintains about the specified tool.

Arguments:

The arguments are:

toolHwnd [required]

The *toolHwnd* and *toolID* arguments are the Rexx object combination that *identify* the tool to be deleted.

toolID [optional]

The *toolHwnd* and *toolID* arguments are the Rexx object combination that *identify* the tool to be deleted.

Return value:

Returns a **`ToolInfo`** object whose attributes reflect the information that this ToolTip maintains about the specified tool. On error, the **`.nil`** object is returned.

Remarks:

The *Tool Identification* section explains exactly how the Rexx object combination is used to specify the tool to be deleted.

Details

Raises syntax errors when incorrect usage is detected.

24.24. hasCurrentTool

```
>>--hasCurrentTool-------------------------------><
```

Tests if this ToolTip has a current tool.

Arguments:

This method has no arguments.

Return value:

Returns true if there is a current tool, false if there is not.

Remarks:

In essence, this convenience method tests if the *getCurrentToolInfo* method will return a **`ToolInfo`** object or the **`.nil`** object.

24.25. hitTestInfo

```
Form 1:

>>--hitTestInfo(--toolInfo--,--point--)----------><

Form 2:

>>--hitTestInfo(--toolInfo--,--x,--y--)----------><

Generic form:

>>--hitTestInfo(--toolInfo--,--ptToTest--)-------><
```

Tests a point to determine whether it is within the bounding rectangle of a tool within the window specified and, if it is, retrieves information about the tool.

Arguments:

The arguments are:

toolInfo [required in / out]

A *ToolInfo* object whose *rexxHwnd* attribute specifies which tool window to test.

If the point tested is within a tool of the tool window, the retrieved tool information is returned in this object. The tool info object should be instantiated using the *forHitTest* class method.

ptToTest [required]

The point to test. The point is specifed in client *coordinates* of the window specified by the *toolInfo* argument.

As indicated by the syntax diagram, the point to test can be specified in two ways. Either as one argument, a *Point* object, or as two arguments, the x coordinate of the point followed by the y coordinate of the point.

Return value:

True if the point being tested is within the window specified, otherwise false.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example gets the current cursor position and then hit tests that position for a push button tool:

```
  mouse = .Mouse~new(self)
  pos = mouse~getCursorPos
  pbTest~screen2client(pos)

  hitTool = .ToolInfo~forHitTest(pbTest)

  say 'Using' pos
  if ttTest~hitTestInfo(hitTool, pos) then do
    say 'Got hit'
    say 'Tool info  hwnd:     ' hitTool~rexxHwnd
    say 'Tool info  id:       ' hitTool~rexxID
    say 'Tool info  text:     ' hitTool~text
    say 'Tool info  flags:    ' hitTool~flags
    say 'Tool info  rect:     ' hitTool~rect
    say 'Tool info  userData:' hitTool~userData
    say 'Tool info  resource:' hitTool~resource
    say
  end
  else do
    say 'NO hit'
    say 'Tool info  hwnd:     ' hitTool~rexxHwnd
  end

/* Output could be for instance:

Got hit
Tool info  hwnd:     a SimpleDialog
Tool info  id:       a Button
Tool info  text:     Press Test to execute the regression suite
Tool info  flags:    IDISHWND SUBCLASS
Tool info  rect:     a Rect (0, 0, 0, 0)
Tool info  userData: The NIL object
Tool info  resource: The NIL object
```

```
*/
```

24.26. manageAtypicalTool

```
>>--manageAtypicalTool(--toolObj--+-----------+--+-----------+--)--------------><
                                  +-,-events--+  +-,-methods-+
```

Initiates the management of a ToolTip tool that is a dialog control. This is for a tool that is not the typical tool used in ooDialog. See the remarks section for details.

The *manageAtypicalTool* initiates a process that monitors every window *message* sent to the *toolObj*, (which is a dialog control.) This allows the process to relay all mouse messages to the tool and to intercept all event notifications sent by the ToolTip to the *toolObj*. Why this is necessary is explained in more detail in the remarks section.

Arguments:

The arguments are:

toolObj [required]

The tool dialog control to be managed.

events [optional]

An array of keywords for the ToolTip events that should invoke a method in the Rexx dialog. The array can not be sparse. Each index in the array should contain the keyword for a ToolTip event. If an event keyword is present at an index, then that event is connected to a method in the Rexx dialog. Case is not significant in the keywords. The valid keywords are:

RELAY	SHOW	LINKCLICK
NEEDTEXT	POP	NORELAY

RELAY

Causes the monitoring process to invoke the connected method in the Rexx dialog for every mouse message sent to the dialog control. The method is invoked *before* the monitoring process relays the message to the ToolTip. By default the method invoked for this keyword is *onRelay*.

NEEDTEXT

Causes the monitoring process to intercept the *NEEDTEXT* event notification sent to the dialog control, and invoke the connected method in the Rexx dialog. The notifcation is prevented from going to the dialog control. By default the method invoked for this keyword is *onNeedText*.

SHOW

Causes the monitoring process to intercept the *SHOW* event notification sent to the dialog control, and invoke the connected method in the Rexx dialog. The notifcation is prevented from going to the dialog control. By default the method invoked for this keyword is *onShow*.

POP

Causes the monitoring process to intercept the *POP* event notification sent to the dialog control and invoke the connected method in the Rexx dialog. The notifcation is prevented from going to the dialog control. By default the method invoked for this keyword is *onPop*.

LINKCLICK

Causes the monitoring process to intercept the *LINKCLICK* event notification sent to the dialog control and invoke the connected method in the Rexx dialog. The notifcation is prevented from going to the dialog control. By default the method invoked for this keyword is *onLinkClick*.

NORELAY

Prevents the monitoring process from automatically forwarding the mouse messages to the ToolTip. There is no method connection associated with this keyword.

methods [optional]

An array of alternative method names for each connected event. If the default method name is not suitable, for whatever reason, then the programmer can supply her own name in the *methods* array. The alternative method name must be at the same index in the *methods* array as the event keyword's index in the *events* array.

E.g., if the keyword SHOW is present at index 2 in the *events* array and the programmer wants to over-ride the default method name of *onShow*, then this can be done by putting the alternative method name at index 2 in the *methods* array. There is no requirement to put any name at index 1. That is, the *methods* array can be sparse.

Return value:

Returns true on success, false on error.

Remarks:

The ooDialog framework provides the *manageAtypicalTool* method to allow more advanced usage of ToolTips. For a ToolTip to work properly, it needs two things. One, it needs to be aware of every mouse message sent to the window that is the tool. Two, it needs to send event notifications to the owner window of the tool.

In the typical use case in ooDialog, the window that the ToolTip sends the notifications to is the dialog, (where they are handled by the Rexx dialog through the event connections.) When the SUBCLASS flag is used for the tool, the ToolTip sets up its own internal process to monitor all the mouse messages sent to the tool window. In this typical use case, there is no need for the *manageAtypicalTool* method.

However there are two basic situations that are not the typical use case. The first is when the SUBCLASS flag can not be used. In this situation there needs to be some way to *relay* the mouse messages to the ToolTip. The problem with mouse messages is that they are sent to the window the mouse is over. If the mouse messages are sent to the dialog window, ooDialog would have no problem. It could intercept the mouse messages through event connections and then relay the message to the ToolTip. However, when the mouse is over a dialog control, the dialog window has no knowledge of the mouse messages.

The other situation is when the dialog window is not sent the ToolTip event notifications. If the event notifications are sent to a dialog control, then there is no way to handle the events through connected methods in the Rexx dialog. Both of these situations can arise when more advanced usage of ToolTips is tried. A common reason why the SUBCLASS flag can not be used is because the application needs to take some action *before* the ToolTip is aware of the mouse message. A common reason why the ToolTip event notifications can not be sent to the dialog window is that the ToolTip was created by and is owned by a dialog control.

There are two example programs provided with the ooDialog distribution that explore the use of the *manageAtypicalTool* method in these two situations. They are in the **`samples\controls\ToolTip`** subdirectories. One program is the **`manageControlTool.rex`** example. The other is the **`customPositionToolTip.rex`** example.

Note that when the *manageAtypicalTool* method starts the monitoring process, it *always* forwards all mouse messages to the ToolTip, even if both optional arguments are ommitted. This is not always desirable. To prevent this, use the NORELAY keyword.

Details

Requires Common Control *Library* version 6.0 or later.

Raises syntax errors when incorrect usage is detected.

Example:

This example sets up a customized ToolTip for a tree-view control. The *manageAtypicalTool* method is used because the customization requires that the application takes action before the mouse move messages are seen by the ToolTip. The RELAY keyword invokes the *onRelay* method in the dialog for every mouse move message. The application examines the mouse messages and determines what action to take. When the method returns, the mouse messages are relayed to the ToolTip.

Note that the third optional argument is an array with no values at index 1 and index 2. The *onShowToolTip* value at index 3 is the alternative method name to invoke for the SHOW event:

```
  tv = self~newTreeView(IDC_TREE)
  rect = tv~clientRect

  tt = self~createToolTip(IDC_TT)

  toolInfo = .ToolInfo~new(tv, 10, '', 'TRANSPARENT', rect)
  tt~addToolEx(toolInfo)

  ...

  tt~manageAtypicalTool(tv, .array~of('RELAY', 'NEEDTEXT', 'SHOW'), .array~of( , ,
 'onShowToolTip'))
```

24.27. newToolRect

```
>>--newToolRect(--rect-,-toolHwnd--+----------+--)--------------><
                                   +-,-toolID-+
```

Sets a new bounding rectangle for a tool.

Arguments:

The arguments are:

rect [required]

A *Rect* object that specifies the new bounding rectangle of the tool.

toolHwnd [required]

The *toolHwnd* and *toolID* arguments are the Rexx object combination that *identify* the tool to be deleted.

toolID [optional]

The *toolHwnd* and *toolID* arguments are the Rexx object combination that *identify* the tool to be deleted.

Return value:

Returns 0, always.

Remarks:

The *Tool Identification* section explains exactly how the Rexx object combination is used to specify the tool the new bounding rectangle is assigned to.

Details

Raises syntax errors when incorrect usage is detected.

24.28. pop

```
>>--pop------------------------------------------><
```

Removes, hides, this ToolTip's display window.

Arguments:

This method has no arguments.

Return value:

Returns 0, always.

24.29. popUp

```
>>--popUp----------------------------------------><
```

Causes this ToolTip to display at the coordinates of the last mouse message.

Arguments:

There are no arguments for this method.

Return value:

Returns 0, always.

Details

Requires Common Control *Library* version 6.0 or later.

Raises syntax errors when incorrect usage is detected.

24.30. setDelayTime

```
>>--setDelayTime(--+---------+--+---------+--)---><
                   +--which--+  +-,-time--+
```

Sets the initial, pop-up, and reshow durations for this ToolTip control.

Arguments:

The arguments are:

which [optional]

Exactly one of the following keywords, case is not significant. This specifies which of the 3, or all of the 3, possible delay times to set. The default if omitted is AUTOMATIC:

AUTOPOP	RESHOW
INITIAL	AUTOMATIC

AUTOPOP

The amount of time the ToolTip window remains visible if the pointer is stationary within a tool's bounding rectangle.

INITIAL

The amount of time the pointer must remain stationary within a tool's bounding rectangle before the ToolTip window appears.

RESHOW

The amount of time it takes for subsequent ToolTip windows to appear as the pointer moves from one tool to another.

AUTOMATIC

Sets all 3 delay times to their default proportions. The autopop time will be ten times the initial time and the reshow time will be one fifth the initial time. When using this keyword, use a positive value for *time* to specify the initial time, in milliseconds. Use a negative *time* to return all three delay times to their default values. This is the default if the *which* argument is omitted.

time [optional]

The time, in milliseconds to set the specified delay time to. A negative time sets the specified time back to its default value. If this argument is omitted, the default is -1.

Return value:

Returns 0, always.

Details

Raises syntax errors when incorrect usage is detected.

24.31. setMargin

```
>>--setMargin(--margins--)-----------------------><
```

Sets the top, left, bottom, and right margins for a ToolTip window. A margin is the distance, in pixels, between the ToolTip window border and the text contained within the ToolTip window.

Arguments:

The single argument is:

margins [required]

A *Rect* object that specifies the margins.

Return value:

Returns 0, always.

Remarks:

The attributes of the returned rectangle do not define a bounding rectangle. For the purpose of this method, the attributes are interpreted as follows:

top:

Distance between top border and top of ToolTip text, in pixels.

left:

Distance between left border and left end of ToolTip text, in pixels.

bottom:

Distance between bottom border and bottom of ToolTip text, in pixels.

right:

Distance between right border and right end of ToolTip text, in pixels.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example sets the margins for a ToolTip to 2 pixels on the sides and 4 pixels on the top and bottom:

```
margins = .Rect~new(2, 4, 2, 4)
toolTip~setMargins(margins)
```

24.32. setMaxTipWidth

```
>>--setMaxTipWidth(--max--)---------------------------------------------><
```

Sets the maximum width, in pixels, for this ToolTip window.

Arguments:

The single argument is:

max [required]

The maximum width for the ToolTip, or -1 to allow any width.

Return value:

Returns the previous maximum width for this ToolTip.

Remarks:

The maximum width value does not indicate a ToolTip window's actual width. Rather, if a ToolTip string exceeds the maximum width, the control breaks the text into multiple lines, using spaces to determine line breaks. If the text cannot be segmented into multiple lines, it is displayed on a single line, which may exceed the maximum ToolTip width.

For instance, if the maximum width is set to 20 and there is a word within the text that is longer than 20, the word is not broken in two. It is displayed on a single line, causing that line to be longer than 20.

The text can contain embedded new line characters and the ToolTip will break the text at the new line indicator.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example creates a tool for a push button with a relatively long string for the tool's text. It then gets the size for a portion of the text in pixels and sets the max tip width to the width of the portion. This causes the ToolTip to create a multi-line display window:

```
tt = self~createToolTip(IDC_TT_MAIN)

text = "Push the 'Test' button to run the current regression test for ooRexx..."
tt~addTool(self~newPushButton(IDC_PB_TEST), text)

s = self~getTextSizePX("Push the 'Test' button")
tt~setMaxTipWidth(s~width)
```

24.33. setTipBkColor

```
>>--setTipBkColor(--color--)---------------------><
```

Sets the background color for this ToolTip window.

Arguments:

The single argument is:

color [required]

The new color for the background of the ToolTip window. The color is expressed as a *COLORREF* value.

Return value:

Returns 0, always.

Remarks:

This method is included for completeness. As is the case for most dialog controls that have set color methods, the dialog control ignores a color change if visual themes are in effect. Most systems on XP and later have visual themes on by default.

Details

Raises syntax errors when incorrect usage is detected.

24.34. setTipTextColor

```
>>--setTipTextColor(--color--)---------------------------------------------><
```

Sets the text color for this ToolTip window.

Arguments:

The single argument is:

color [required]

The new color for the text of the ToolTip window. The color is expressed as a *COLORREF* value.

Return value:

Returns 0, always.

Remarks:

This method is included for completeness. As is the case for most dialog controls that have set color methods, the dialog control ignores a color change if visual themes are in effect. Most systems on XP and later have visual themes on by default.

Details

Raises syntax errors when incorrect usage is detected.

24.35. setTitle

```
>>--setTitle(--title--+---------+--)-------------><
                      +-,-icon--+
```

Adds a title string and optionally an icon image to this ToolTip.

Arguments:

The arguments are:

title [required]

The title for this ToolTip.

icon [optional]

Either an icon *Image* object, or one of the following keywords to use a system icon. Case is not significant in the keywords. The default if the keyword is omitted is NONE:

NONE	INFO	WARNINGLARGE
ERROR	INFOLARGE	
ERRORLARGE	WARNING	

NONE

No icon.

ERROR

Error icon.

ERRORLARGE

Large warning icon.

INFO

Info icon.

INFOLARGE

Large info icon.

WARNING

Warning icon.

WARNINGLARGE

Large warning icon.

Return value:

True on success, false on error.

Remarks:

The *title* string can not be longer than 99 characters. If an *Image* object is used, the image must actually be an icon. The operating system will not display a bitmap when an icon is called for.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example creates a balloon ToolTip with a title and an icon;

```
icon = .Image~getImage("musicNote.ico", .Image~toID(IMAGE_ICON), .size~new(16, 16))

tt = self~createToolTip(IDC_TT_BALLOON, 'BALLOON CLOSE')
tt~setTitle("Important Message", icon)
```

24.36. setToolInfo

```
>>--setToolInfo(--toolInfo--)--------------------><
```

Sets the information that this ToolTip control maintains for a tool.

Arguments:

The single argument is:

toolInfo [required]

A *ToolInfo* object whose attributes specify the information this ToolTip should use for a tool. See the remarks.

Return value:

Returns 0, always.

Remarks:

Some internal properties of a tool are established when the tool is created, and are not recomputed by the ToolTip when the *setToolInfo* method is invoked. If a **`ToolInfo`** object is simply instantiated, its attributes assigned values, and is then passed to the ToolTip control through the *setToolInfo* method, these properties may be lost.

Instead, the programmer should first request the tool's current **`ToolInfo`** object by using the *getToolInfo* method. Then, modify the attributes of this object as needed and pass it back to the ToolTip control using the *setToolInfo* method.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example modifies the text, bounding rectangle, and flags of a tool:

```
toolInfo = ttEsc~getToolInfo(self~newPushButton(IDCANCEL))

toolInfo~text  = "Only push the cancel button if you want, or need, to end the dialog"
toolInfo~rect  = .Rect~new(3, 2, 44, 11)
toolInfo~flags = toolInfo~flags 'CENTERTIP'
```

```
ttEsc~setToolInfo(toolInfo)
```

24.37. setWindowTheme

```
>>--setWindowTheme(--style--)--------------------><
```

Sets the visual style of a ToolTip control.

Arguments:

The single argument is:

style [required]

A string specifying the visual style.

Return value:

Returns 0, always.

Remarks:

This method over-rides the *DialogControl* class's *setWindowTheme* method because the implementation for a tool tip is different than for the general dialog control case.

This method is included for completeness. The Windows *documentation* is very sparse on this topic. It simply says a string specifying the style, but doesn't say what styles are available. Currently, the ooDialog developers are not certain what string to use to get this method to work.

Details

Requires Common Control *Library* version 6.0 or later.

Raises syntax errors when incorrect usage is detected.

24.38. trackActivate

```
>>--trackActivate(--toolHwnd--+----------+--+------------+--)--><
                              +-,-toolID-+  +-,-activate--+
```

Activates or deactivates a tracking ToolTip.

Arguments:

The arguments are:

toolHwnd [required]

The *toolHwnd* and *toolID* arguments are the Rexx object combination that *identify* the tool to be deleted.

toolID [optional]

The *toolHwnd* and *toolID* arguments are the Rexx object combination that *identify* the tool to be deleted.

activate

Must be true or false, use true to activate the ToolTip, false to deactivate it. If omitted, the default is true.

Return value:

Returns 0, always.

Remarks:

The *Tool Identification* section explains exactly how the Rexx object combination is used to specify the tool to be deleted.

Tracking ToolTips must be manually activated and deactivated by using the *trackActivate* method. Activation or deactivation also shows or hides the ToolTip, respectively.

While a tracking ToolTip is active, the application must specify the location of the ToolTip by invoking the *trackPosition* method.

Details

Raises syntax errors when incorrect usage is detected.

24.39. trackPosition

```
Form 1:

>>--trackPosition(--point--)---------------------><

Form 2:

>>--trackPosition(--x,--y--)---------------------><

Generic form:

>>--trackPosition(--pt--)------------------------><
```

Sets the position of a tracking ToolTip.

Arguments:

The arguments are:

pt [required]

The (x, y) coordinates of the position to set. As the syntax diagram indicates, the coordinates can be specified either as a single *Point* object, or as two arguments, a separate x and a y coordinate.

Return value:

Returns 0, always.

Remarks:

Tracking ToolTips change position on the screen dynamically. While a tracking ToolTip is active, the application must specify the location of the ToolTip by invoking the *trackPosition* method.

The ToolTip control chooses where to display the ToolTip window based on the coordinates provided through the *trackPosition* method. This causes the ToolTip window to appear beside the tool to which it corresponds. To have ToolTip windows displayed at specific coordinates, include the ABSOLUTE keyword in the *flags* argument when adding the tool through the *addTool*, *addToolEx*, or *addToolRect* methods.

Details

Raises syntax errors when incorrect usage is detected.

24.40. update

```
>>--update---------------------------------------><
```

Forces the current tool to be redrawn.

Arguments:

There are no arguments for this method.

Return value:

Returns 0, always.

24.41. updateTipText

```
>>--updateTipText(--toolInfo--)------------------><
```

Sets the ToolTip text for a tool.

Arguments:

The arguments are:

toolInfo [required]

A *ToolInfo* object that specifies the tool and the text for the update.

Return value:

Returns 0, always.

Remarks:

The best way to use this method is to instantiate a new **`ToolInfo`** object through the *forID* method and then set the *text* attribute of the object to the new text desired.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example updates the text for a tool:

```
  toolInfo = .ToolInfo~forID(self~newPushButton(IDCANCEL))
  toolInfo~text  = "Only push the cancel button if you want, or need, to end the dialog"

  toolTip~updateTipText(toolInfo)
```

24.42. ToolInfo Class

A *ToolInfo* object contains information about a tool in a *ToolTip* control. The attributes of the **`ToolInfo`** object reflect the information that defines a specific tool in the ToolTip control.

ToolInfo objects are used in some of the methods of the **`ToolTip`** class to provide information to the ToolTip control, and / or, to receive information from the ToolTip control.

24.42.1. Method Table

The following table lists the class and instance methods of the **`ToolInfo`** class:

Table 24.3. ToolInfo Class Method Reference

Method	Description
Class Methods	**Class Methods**
forHitTest	Returns a new **`ToolInfo`** object to be used in the *hitTestInfo* method.
forID	Returns a new **`ToolInfo`** object to be used for tool *identification.*
new	Instantiates a new **`ToolInfo`** object.
Attribute Methods	**Attribute Methods**
flags	Reflects the flags that define the style and options of the tool.
rect	Reflects the bounding rectangle coordinates of the tool when the tool is an application-defined rectangular area within a window's client area.
resource	Reserved for future enhancement.
rexxHwnd	Reflects the Rexx object from which the first part of the 2-part system *identification* for a tool is derived.
rexxID	Reflects the Rexx object from which the second part of the 2-part system *identification* for a tool is derived.
text	Reflects the text set for the tool.
userData	Reflects the application-defined user data value for the tool.

24.42.2. forHitTest (Class Method)

```
>>--forHitTest(--hwndObj--)----------------------><
```

Returns a new **`ToolInfo`** object to be used in the *hitTestInfo* method.

Arguments:

The single argument is:

hwndObj [required]

The Rexx tool or Rexx object that contains a tool that is going to be hit tested. For example, if the tool to be tested is a rectangular area, *hwndObj* should be the Rexx object that represents the window containing the rectangle. If the tool is a dialog control, then *hwndObj* should be the Rexx object that represents the dialog control.

Return value:

Returns a newly instantiated **`ToolInfo`** object designed to be used as the argument to the *hitTest* method.

Remarks:

The new **`ToolInfo`** object is especially constructed to be of use in the *hitTestInfo* method. To be explicit, the **`ToolInfo`** object returned is initialized to an empty state and the normal tool identifiers are not set. The returned **`ToolInfo`** object is not suitable to be used in any other methods that required a **`ToolInfo`** object, except the *hitTest* method.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example comes from an application that has a number of rectangular tools in a dialog. An arbitrary point is hit tested and if the hit succeeds, the filled in tool information is printed to the screen:

```
  hitTool = .ToolInfo~forHitTest(self)

  say 'Using' pos

  if tt~hitTest(hitTool, pos) then do
    say 'Got hit'
    say 'Tool info  hwnd:     ' hitTool~rexxHwnd
    say 'Tool info  id:       ' hitTool~rexxID
    say 'Tool info  text:     ' hitTool~text
    say 'Tool info  flags:    ' hitTool~flags
    say 'Tool info  rect:     ' hitTool~rect
    say 'Tool info  userData:' hitTool~userData
    say 'Tool info  resource:' hitTool~resource
    say
  end
  else do
    say 'NO hit'
  end

/*  Output might be for instance:

Using a Point (220, 101)
Got hit
Tool info  hwnd:     a SimpleDialog
Tool info  id:       777
Tool info  text:     Over main dialog, bottom right quadrant
Tool info  flags:    SUBCLASS TRANSPARENT
Tool info  rect:     a Rect (194, 101, 386, 200)
Tool info  userData: The NIL object
Tool info  resource: The NIL object

*/

/* Output from a miss:

Using a Point (220, 301)
NO hit

*/
```

24.42.3. forID (Class Method)

```
>>--forID(--toolHwnd--+----------+--)------------><
                      +-,-toolID-+
```

Returns a new **`ToolInfo`** object to be used for tool *identification.*

Arguments:

The arguments are:

toolHwnd [required]

The *toolHwnd* and *toolID* arguments are the Rexx object combination that *identify* a tool to the operating system.

toolID [optional]

The *toolHwnd* and *toolID* arguments are the Rexx object combination that *identify* the tool to the operaring system.

Return value:

Returns a newly instantiated **ToolInfo** object that can be used to identify a specific tool contained within a ToolTip.

Remarks:

Many of the **ToolTip** methods require that the tool the method pertains to be identified. The **ToolInfo** object returned from *forID* is useful for those method. In addition, some methods return information by filling in the attributes of a **ToolInfo** object that specifies a tool. This is another good use of the *forID* method.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example instantiates a new **ToolInfo** object using the *forID* method and then uses the object to change the ToolTip text for a push button tool:

```
 toolInfo = .ToolInfo~forID(self~newPushButton(IDC_PB_TEST))
 toolInfo~text = "Push the 'Test' button when you are ready to begin the test suite for
the final release."

 toolTip~setMaxTipWidth(140)
 toolTip~updateTipText(toolInfo)
```

24.42.4. new (Class Method)

```
>>--new(-toolHwnd-+----------+-+-------+-+--------+-+-----+-+---------+-+-------+-)--><
                  +-,-toolID-+ +-,-txt-+ +-,-flgs-+ +-,-r-+ +-,-uData-+ +-,-rsv-+
```

Instantiates a new **ToolInfo** object.

Arguments:

The arguments are:

toolHwnd [required]

The *toolHwnd* and *toolID* arguments are the Rexx object combination that *identify* a tool to the operating system.

toolID [optional]

The *toolHwnd* and *toolID* arguments are the Rexx object combination that *identify* the tool to the operaring system.

txt [optional]

Text for the tool. If omitted, or the empty string, or the string: TEXTCALLBACK, then the ToolTip sends the *NEEDTEXT* notification and the program supplies the text.

The length of the text must be less than 1024 characters, which includes any possible end of line (0x0D0A) sequences.

flags [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant.

ABSOLUTE	PARSELINKS	TRACK
CENTERTIP	RTLREADING	TRANSPARENT
IDISHWND	SUBCLASS	

ABSOLUTE

Positions the ToolTip window at the exact same coordinates specified by the *trackPosition* method. Without this flag the ToolTip control chooses where to display the ToolTip window based on the coordinates specified, which places the ToolTip close to the tool. This flag must be used with the TRACK flag.

CENTERTIP

Centers the ToolTip window below the tool.

IDISHWND

Indicates that the ID part of the tool *identification* is the window handle to the tool. In general, the ooDialog framework will set this flag correctly by using knowledge it gains from parsing the *toolHwnd* and *toolID* arguments. If the flag is required, the framework will set it, if the flag should not be used, the framework will remove it if it is set.

PARSELINKS

Requires Common Control *Library* version 6.0 or later.

Indicates that links in the ToolTip text should be parsed.

RTLREADING

Indicates that the ToolTip text will be displayed in the opposite direction to the text in the parent window.

SUBCLASS

Indicates that the ToolTip control should subclass the tool's window to intercept messages, such as WM_MOUSEMOVE. If this flag is not set, some means must be used so that the mouse messages are forwarded to the ToolTip control, for instance by using the *manageAtypicalTool* method.

TRACK

Positions the ToolTip window next to the tool to which it corresponds and moves the window according to coordinates supplied by the *trackPosition* method. The programmer must activate this type of tool using the *trackActivate* method.

TRANSPARENT

Causes the ToolTip control to forward mouse event messages to the parent window. This is limited to mouse events that occur within the bounds of the ToolTip window.

r [optional]

A *Rect* object that specifies the bounding rectangle coordinates of the tool. The coordinates are in client *coordinates* of the tool.

uData [optional]

A user data value to be associated with the tool. This can be any Rexx object the programmer wants. Note that the value is associated with the tool, not the tool tip.

rsv [optional]

Reserved for future enhancement. This argument is completely ignored in the current implmentation of ooDialog. Future versions may be enhance to use this argument to specify a *ResourceImage* object containing binary resources for the tool.

Return value:

Returns a newly instantiated **ToolInfo** object.

Remarks:

Except for the IDISHWND flag, the returned **ToolInfo** object is constructed exactly as the programmer specifies. This makes the object ideal for use in the *addToolEx* method for those cases where the convenience *addTool* and *addToolRect* methods are not suitable.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example uses the *new* method to instantiate a **ToolInfo** object for an implementation of custom ToolTips for a *TreeView*:

```
tv = self~newTreeView(IDC_TREE)
ret = tv~setItemHeight(20)
tt = self~createToolTip(IDC_TT)

rect = tv~clientRect

toolInfo = .ToolInfo~new(tv, ID_TOOL_TV, '', 'TRANSPARENT', rect, helperObj)
tt~addToolEx(toolInfo)

tt~setMaxTipWidth(250)

self~populateTree(tv)
tt~manageAtypicalTool(tv, .array~of('RELAY', 'NEEDTEXT', 'SHOW'))
```

24.42.5. flags (Attribute)

```
>>--flags-----------------------------------------------------><

>>--flags = varName-------------------------------------------><
```

Reflects the flags that define the style and options of the tool.

flags get:

Returns a string of 0 or more keywords separated by a single space that specify the flags set for this tool. If no flags are set, the empty string is returned. See the remarks for the keyword values and meanings.

flags set:

Sets the flags for this tool to those specified by a string of 0 or more space separated keywords. Case is not significant. See the remarks for the possible keywords and their meaning.

Remarks:

The following keywords specify the individual flags that define the style and options for a ToolTip tool. The keywords are case insensitive.

ABSOLUTE	PARSELINKS	TRACK

CENTERTIP	RTLREADING	TRANSPARENT
IDISHWND	SUBCLASS	

ABSOLUTE
: Indicates that the ToolTip should position its window at the exact same coordinates specified by the *trackPosition* method. Without this flag the ToolTip control chooses where to display the ToolTip window based on the coordinates specified, which places the ToolTip close to the tool. This flag must be used with the TRACK flag.

CENTERTIP
: Indicates that the ToolTip should centers its window below the tool.

IDISHWND
: Indicates that the ID part of the tool *identification* is the window handle if the tool. If the programmer sets the *flags* attribute, it becomes the programmer's responsibility to set this flag correctly. Failure to do so will result in unpredictable behaviour.

PARSELINKS
: Requires Common Control *Library* version 6.0 or later.

  Indicates that links in the ToolTip text should be parsed.

RTLREADING
: Indicates that the ToolTip text will be displayed in the opposite direction to the text in the parent window.

SUBCLASS
: Indicates that the ToolTip control should subclass the tool's window to intercept messages, such as WM_MOUSEMOVE. If this flag is not set, some means must be used so that the mouse messages are forwarded to the ToolTip control.

TRACK
: Indicates that the ToolTip should position its window next to the tool to which it corresponds and move the window according to coordinates supplied by the *trackPosition* method. The programmer must activate this type of tool using the *trackActivate* method.

TRANSPARENT
: Causes the ToolTip control to forward mouse event messages to the parent window. This is limited to mouse events that occur within the bounds of the ToolTip window.

Details

Raises syntax errors when incorrect usage is detected.

24.42.6. rect (Attribute)

```
>>--rect-----------------------------------------------------><

>>--rect = varName-------------------------------------------><
```

Reflects the bounding rectangle coordinates of the tool when the tool is an application-defined rectangular area within a window's client area.

rect get:

Returns the bounding rectangle as a *Rect* object.

rect set:

Assign a **`Rect`** object to the *rect* attribute to define the tool as a rectangular area within a window.

Remarks:

When the tool is not an application-defined rectangular area, the value of the *rect* attribute is a **`rect`** object whose coordinates are all 0. When the *flags* attribute includes the IDISHWND keyword, the *rect* attriubte is ignored by the ToolTip.

Details

Raises syntax errors when incorrect usage is detected.

24.42.7. resource (Attribute)

```
>>--resource-----------------------------------------------------><

>>--resource = varName-------------------------------------------><
```

Reserved for future enhancement.

resource get:

Returns the **`.nil`** object, always.

resource set:

This attribute can not be set.

Remarks:

If the *ResourceImage* class is enhanced to work with string resources (a good possibility) then this attribute will be the **`ResourceImage`** that contains the string resource to use for the text of the tool.

Details

Raises syntax errors when incorrect usage is detected.

24.42.8. rexxHwnd (Attribute)

```
>>--rexxHwnd-----------------------------------------------------><

>>--rexxHwnd = varName-------------------------------------------><
```

Reflects the Rexx object from which the first part of the 2-part system *identification* for a tool is derived.

rexxHwnd get:

Returns the Rexx object that is used as part of the basis for the tool identification. This will always be either a dialog object or a dialog control object.

rexxHwnd set:

The Rexx programmer can not set this attribute. It is set by the ooDialog framework when the **`ToolInfo`** object is instantiated and can not be changed after that.

Details

Raises syntax errors when incorrect usage is detected.

24.42.9. rexxID (Attribute)

```
>>--rexxID-----------------------------------------------------><

>>--rexxID = varName-------------------------------------------><
```

Reflects the Rexx object from which the second part of the 2-part system *identification* for a tool is derived.

rexxID get:

Returns the Rexx object used as the ID portion of the tool identifier. This will always be either a dialog control object or a non-negative whole number. (Recall that in ooRexx all numbers are objects.)

rexxID set:

The Rexx programmer can not set this attribute. It is set by the ooDialog framework when the **`ToolInfo`** object is instantiated and can not be changed after that

Details

Raises syntax errors when incorrect usage is detected.

24.42.10. text (Attribute)

```
>>--text-------------------------------------------------------><

>>--text = varName---------------------------------------------><
```

Reflects the text set for the tool.

text get:

Returns the text for the tool, the empty string if the text is not set, or the string *TextCallBack* if the tool is set to use the call back feature. See the remarks for details on the call back feature.

text set:

Assign this attribute the text string the ToolTip should display in its window for this tool. To set the ToolTip to use its call back feature, either assign the string *TEXTCALLBACK*, case is not significant, or assign the empty string.

Remarks:

The ToolTip provides a call back feature that can be used rather than setting a static text string for the tool. With this feature, the ToolTip sends the *NEEDTEXT* notification and the program supplies the text supplies the text for the tool by responding to the notification. To use this feature, the programmer should set the *text* attribute to either the string, *TextCallBack* ro the empty string.

The length of the text must be less than 1024 characters, which includes any possible end of line (0x0D0A) sequences. Do not set the *text* attriubte to a string longer than 1023 characters.

Details

Raises syntax errors when incorrect usage is detected.

24.42.11. userData (Attribute)

```
>>--userData-----------------------------------------------------><

>>--userData = varName-------------------------------------------><
```

Reflects the application-defined user data value for the tool.

userData get:

Returns the user data set for the tool, or the **`.nil`** object is no user data has been assigned.

userData set:

The programmer can assign a user data value to the tool. This can be any Rexx object desired.

Details

Raises syntax errors when incorrect usage is detected.

Trackbar Controls

A trackbar is a window that contains a slider and optional tick marks. Prior to the effort to *unify* the ooDialog class and method names, a trackbar was called a slider control. This was a misnomer, the slider is a part of the track bar. Trackbars are now called by their correct name in this documentation.

Trackbars are useful if you want the user to select a specific value or a set of consecutive values within a specific range. For example, you might use a trackbar to allow the user to set the repeat rate of the keyboard by moving the slider in the trackbar to a given tick mark.

When the user moves the slider in the trackbar, using either the mouse or the direction keys, the trackbar sends notification messages to its parent dialog to indicate the change.

The slider in a trackbar moves in increments that you specify when you create it. For example, if you specify that the trackbar should have a range from 0 to 10, the slider can occupy only eleven positions: a position at the left side of the slider and one position for each increment in the range. Typically, each of these positions is identified by a tick mark.

After you have created a trackbar, you can use its methods to set and retrieve its properties. Refer to the **`propdemo.rex`** example program included in the ooRexx distribution under the **`samples`** subdirectory for some help in getting started using a track bar.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with trackbar controls:

Instantiation:

Use the *newTrackBar* method of the *dialog* object to retrieve a new TrackBar object.

Dynamic Definition:

To dynamically define a trackbar in the dialog template of a *UserDialog* class, use the *createTrackBar* method.

Event Notification

To connect the *event* notifications sent by the underlying trackbar control to a method in the Rexx dialog object use the *connectTrackBarEvent* method.

25.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with **`TrackBar`** objects, including the pertinent methods from other classes:

Table 25.1. Trackbar Instance Methods

Method	Description
Useful	**External Methods**
newTrackBar	Returns a **`Trackbar`** object for the control with the specified ID.
createTrackBar	Creates a trackbar control in the dialog template of a *UserDialog*
connectTrackBarEvent	Connects trackbar event notifications to a method in the Rexx dialog object
Constant	**Methods**
constants	The **`TrackBar`** class provides a number of constant values used to decode the reason for *MOVE* event notification.
Instance	**Methods**
clearSelRange	Clears the current selection range in a trackbar.

Method	Description
clearTicks	Removes the current tick marks from a trackbar.
countTicks	Retrieves the number of tick marks in a trackbar,
getLineStep	Retrieves the number of logical positions that the slider in a trackbar moves in response to keyboard input from the arrow keys.
getPageStep	Retrieves the number of logical positions that the slider in a trackbar moves in response to page up or down keys, or mouse clicks.
getTick	Retrieves the logical position of a tick mark in a trackbar.
initRange	Sets the minimum and maximum positions of the trackbar and redraws the trackbar, if requested.
initSelRange	Sets the starting and ending logical positions for the current selection range in a trackbar.
pos	Retrieves the current logical position of the slider in a trackbar.
pos=	Sets the logical position of the slider in a trackbar, however the trackbar is not redrawn.
range	Retrieves the minimum and maximum positions of the trackbar.
selRange	Retrieves the starting position of the current selection range in a trackbar.
setLineStep	Sets the number of logical positions that the slider in a trackbar moves in response to keyboard input from the arrow keys.
setMax	Sets the maximum logical position for a trackbar and redraws the trackbar, if requested.
setMin	Sets the minimum logical position for a trackbar and redraws the trackbar, if requested.
setPageStep	Sets the number of logical positions that the slider in a trackbar moves in response to page up or down keys, or mouse clicks.
setPos	Sets the new logical position of the slider in a trackbar and redraws the trackbar, if requested.
setSelEnd	Sets the logical ending position for the current selection range in a trackbar.
setSelStart	Sets the starting logical position for the current selection range in a trackbar.
setTickAt	Sets a tick mark at the specified logical position in the trackbar.
setTickFrequency	Sets the interval frequency for tick marks in a trackbar

25.2. TrackBar Constants

A trackbar notifies the owner dialog of user interaction by sending an event notification to the dialog. This *MOVE* event can be connected through the *connectTrackBarEvent* method. Part of the information sent the event handler is the reason for the notifcation. The following table lists the trackbar notification constants and the events (virtual key codes or mouse events) that cause the notifications to be sent.

Table 25.2. Trackbar Constants Table

Constant	Event Notification Reason
UP	The virtual LEFT or UP keys
DOWN	The virtual RIGHT or DOWN keys.

Constant	Event Notification Reason
PAGEUP	The PRIOR virtual key (the user clicked the channel above or to the left of the slider.)
PAGEDOWN	The virtual NEXT key (the user clicked the channel below or to the right of the slider.
POSITION	Left mouse button up, following a THUMBTRACK notification message.
DRAG	Slider movement (the user dragged the slider with the mouse.)
TOP	The RIGHT or DOWN virtual keys'
BOTTOM	The virtual END key.
ENDTRACK	KEYUP (the user released a key that sent a relevant virtual key code.)

25.3. newTrackBar (dialog object method)

Track bar objects can not be instantiated by the programmer from Rexx code. Rather a trackbar object is obtained by using the *newTrackBar* method of the *dialog* object. The syntax is:

```
>>-newTrackBar(--id--)---------------------------><
```

25.4. createTrackBar (UserDialog method)

A trackbar control can be added to the dialog template for a *UserDialog* dialog through the *createTrackbar*() method. The basic syntax is:

```
>>--createTrackBar(-id-,--x-,--y-,--cx-,--cy-+---------+--+-----------+--)----><
                                             +-,-style-+  +-,-attrName-+
```

25.5. connectTrackBarEvent (dialog object method)

To connect event notifications from an trackbar control use the *connectTrackBarEvent*() method of the *dialog* object. The basic syntax is:

```
>>--connectTrackBarEvent(--id--,--event--+---------------+--)---><
                                         +-,--methodName-+
```

25.6. clearSelRange

```
>>--clearSelRange(--+----------+--)--------------><
                    +--redraw--+
```

The clearSelRange method clears the current selection range in a slider.

Arguments:

The only argument is:

redraw [optional]

The redraw flag. If **`.true`**, the trackbar is redrawn after the selection is cleared. If **`.false`** the trackbar is not redrawn. The default is **`.true`**.

Return value:

0.

25.7. clearTicks

```
>>--clearTicks(--+----------+--)-----------------><
                 +--redraw--+
```

The clearTicks method removes the current tick marks from a slider. It does not, however, remove the first and last tick marks because they are created automatically by the slider.

Arguments:

The arguments are:

redraw [optional]

The redraw flag, if **`.true`** the trackbar is redrawn after the tick marks are removed. If **`.false`** the trackbar is not redrawn. The default is **`.true`**.

Return value:

0.

25.8. countTicks

```
>>--countTicks-----------------------------------><
```

The countTicks method retrieves the number of tick marks in a trackbar, including the first and last tick marks, which are created automatically by the slider.

Return value:

The number of tick marks.

25.9. getLineStep

```
>>--getLineStep----------------------------------><
```

The getLineStep method retrieves the number of logical positions that the trackbar moves in response to keyboard input from the arrow keys, such as the Right arrow or the Down arrow keys. The logical positions are the integer increments within the minimum and maximum positions of the slider.

Return value:

The line size for the slider.

25.10. getPageStep

```
>>--getPageStep----------------------------------><
```

The getPageStep method retrieves the number of logical positions that the trackbar moves in response to keyboard input, such as the PageUp or PageDown keys, or mouse input, such as clicks in the trackbar's channel. The logical positions are the integer increments within the minimum and maximum positions of the slider.

Return value:

The page size for the slider.

25.11. getTick

```
>>--getTick(--tic--)-----------------------------><
```

The getTick method retrieves the logical position of a tick mark in a slider. A valid position is an integer value within the minimum and maximum positions of the slider.

Arguments:

The only argument is:

tic

A zero-based index identifying a tick mark. A valid index is in the range of 0 to 2 ticks less than the tick count returned by the countTicks method (see *countTicks*).

Return value:

The logical position of the specified tick mark, or -1 if you did not specify a valid index for *tic*.

25.12. initRange

```
>>--initRange(--+-----+--+-------+--+-----------+--)-----------><
                +-min-+  +-,-max-+  +-,-redraw--+
```

The initRange method sets the minimum and maximum positions of the trackbar and redraws the trackbar, if required.

Arguments:

The arguments are:

min [optional]

The minimum position of the slider. The default is 0.

max [optional]

The maximum position of the slider. The default is 100.

redraw [optional]

The redraw flag. If you specify **`.true`**, the trackbar is redrawn after the range is set. If you specify **`.false`** the trackbar is not redrawn. The default is false.

Return value:

0

The range was set.

-1

The minimum you specified is greater than the maximum.

Example:

The following example sets the range, the line step, the page step, and the tick frequency of the slider:

```
::method initDialog
```

```
  curSL = self~newTrackBar("IDC_1")
  if curSL \= .Nil then do
    curSL~initRange(0,100)
    curSL~setLineStep(1)
    curSL~setPageStep(10)
    curSL~setTickFrequency(10)
  end
```

25.13. initSelRange

```
>>--initSelRange(--+-----+--,--+-----+--,--+--------+--)--------><
                   +-min-+     +-max-+     +-redraw-+
```

The initSelRange method sets the starting and ending logical positions for the current selection range in a slider. It is ignored if the trackbar does not have a selection range.

Arguments:

The arguments are:

min

The logical starting position of the selection range. The default is 0.

max

The logical ending position of the selection range. If you omit this argument, the maximum position of the trackbar's range is assumed.

redraw

The redraw flag. If you specify **`.true`**, the trackbar is redrawn after the selection range is set. If you specify **`.false`** the selection range is set but the trackbar is not redrawn. The default is **`.false`**.

Return value:

-1

The minimum you specified is greater than the maximum.

0

In all other cases.

25.14. pos

```
>>--pos------------------------------------------><
```

The pos method retrieves the current logical position of the slider.

Return value:

The current logical position of the slider.

Example:

The following example displays the current trackbar position:

```
::method displayPos
  ctrl=self~newTrackBar("IDC_1")
  pos = ctrl~pos
  say pos
```

25.15. pos=

```
>>--pos=value------------------------------------><
```

The pos= method sets the new logical position of the trackbar and redraws the slider.

Arguments:

The only argument is:

value

The new logical position. A valid position is an integer value within the range of the minimum and maximum positions of the slider. If you specify a value outside this range, the position is set to the maximum or minimum position.

25.16. range

```
>>--range----------------------------------------><
```

The range method retrieves the minimum and maximum positions of the slider.

Return value:

The minimum and maximum positions of the trackbar, separated by a blank.

Example:

The following example displays the range of a slider:

```
::method displayRange
  ctrl=self~newTrackBar("IDC_1")
  range = ctrl~range
  parse var range min max
  say min max
```

25.17. selRange

```
>>--selRange-------------------------------------><
```

The selRange method retrieves the starting position of the current selection range in a slider.

Return value:

The starting and ending positions of the current selection range, separated by a blank.

25.18. setLineStep

```
>>--setLineStep(--step--)------------------------><
```

The setLineStep method sets the number of logical positions that the trackbar moves in response to keyboard input from the arrow keys, such as the Right arrow or the Down arrow keys. The logical positions are the integer increments within the minimum and maximum positions of the slider.

Arguments:

The only argument is:

step

The new line size.

Return value:

The previous line size, or -1 if you omit *step*.

Example:

The following example sets the range, the line step, the page step, and the tick frequency of the slider:

```
::method initDialog
  curSL = self~newTrackBar("IDC_1")
  if curSL \= .Nil then do
    curSL~initRange(0,100)
    curSL~setLineStep(1)
    curSL~setPageStep(10)
    curSL~setTickFrequency(10)
  end
```

25.19. setMax

```
>>--setMax(--max--+-----------+--)---------------><
                  +-,-redraw--+
```

The setMax method sets the maximum logical position for a trackbar and redraws the trackbar, if required.

Arguments:

The arguments are:

min

The maximum position of the slider.

redraw [optional]

The redraw flag. If you specify **`.true`** the trackbar is redrawn after the maximum position is set. If you specify **`.false`** the trackbar is not redrawn. The default is **`.true`**.

Return value:

0

The maximum position was set successfully.

-1

You omitted *min*.

25.20. setMin

```
>>--setMin(--min--+-----------+--)---------------><
```

```
                 +-,-redraw--+
```

The setMin method sets the minimum logical position for a trackbar and redraws the trackbar, if required.

Arguments:

The arguments are:

min

The minimum position of the slider.

redraw [optional]

The redraw flag. If you specify **`.true`** the trackbar is redrawn after the minimum position is set. If you specify **`.false`** the trackbar is not redrawn. The default is **`.true`**.

Return value:

0

The minimum position was set.

-1

You omitted *min*.

25.21. setPageStep

```
>>--setPageStep(--step--)------------------------><
```

The setPageStep method sets the number of logical positions that the trackbar moves in response to keyboard input, such as the PageUp or PageDown keys, or mouse input, such as clicks in the trackbar's channel. The logical positions are the integer increments within the minimum and maximum positions of the slider.

Arguments:

The only argument is:

step

The new page size.

Return value:

The previous page size.

Example:

The following example sets the range, the line step, the page step, and the tick frequency of the slider:

```
::method initDialog
  curSL = self~newTrackBar("IDC_1")
  if curSL \= .Nil then do
    curSL~initRange(0,100)
    curSL~setLineStep(1)
    curSL~setPageStep(10)
    curSL~setTickFrequency(10)
  end
```

25.22. setPos

```
>>--setPos(--pos--+-----------+--)---------------><
                  +-,-redraw--+
```

The setPos method sets the new logical position of the trackbar and redraws the trackbar if required.

Arguments:

The arguments are:

pos

The new logical position. A valid position is an integer value within the range of the minimum and maximum positions of the slider. If you specify a value outside this range, the position is set to the maximum or minimum position.

redraw [optional]

The redraw flag. If you specify **.true**, the control is redrawn with the trackbar at the position given by *pos*. If you specify **.false** the control is not redrawn. However, the new position is set regardless of the redraw argument. The default is **.false**.

Example:

The following example sets the trackbar to the maximum position, with the range of the trackbar already been set to 0 to 100 using the setRange method:

```
::method setToMax
  ctrl=self~newTrackBar("IDC_1")
  ctrl~setPos(100,1)
```

25.23. setSelEnd

```
>>--setSelEnd(--max--+----------+--)-------------><
                     +-,-redraw-+
```

The setSelEnd method sets the logical ending position for the current selection range in a slider. It is ignored if the trackbar does not have a selection range.

Arguments:

The arguments are:

min

The logical ending position of the selection range.

redraw [optional]

The redraw flag. If you specify **.true** the trackbar is redrawn after the ending position has been set. If you specify **.false** the ending position is set but the trackbar is not redrawn. The default is **.true**.

Return value:

-1

You omitted *max*.

0

In all other cases.

25.24. setSelStart

```
>>--setSelStart(--min--+----------+--)----------><
                       +-,-redraw-+
```

The setSelStart method sets the starting logical position for the current selection range in a slider. It is ignored if the trackbar does not have a selection range.

Arguments:

The arguments are:

min

The logical starting position of the selection range.

redraw [optional]

The redraw flag. If you specify **`.true`** the trackbar is redrawn after the starting position has been set. If you specify **`.false`** the starting position is set but the trackbar is not redrawn. The default is **`.true`**.

Return value:

-1

You omitted *min*.

0

In all other cases.

25.25. setTickAt

```
>>--setTickAt(--pos--)--------------------------><
```

The setTickAt method sets a tick mark at the specified logical position in the slider.

Arguments:

The only argument is:

pos

An integer value within the minimum and maximum positions of the slider.

Return value:

0

The tick mark was set.

-1

You omitted *pos*.

1

For all other cases.

25.26. setTickFrequency

```
>>--setTickFrequency(--freq--)------------------><
```

The *setTickFrequency* method sets the interval frequency for tick marks in a trackbar. For example, if you set the frequency to 2, a tick mark is displayed for every other increment in the trackbar's range. By default the frequency is 1.

Arguments:

The single argument is:

freq [required]

The frequency of the tick marks. By default the trackbar sets its frequency to 1, that is, every increment in the range is associated with a tick mark.

Return value:

Returns 0, always.

Example:

The following example sets the range, the line step, the page step, and the tick frequency of the trackbar:

```
::method initDialog
  curSL = self~newTrackBar("IDC_1")
  if curSL \= .nil then do
    curSL~initRange(0,100)
    curSL~setLineStep(1)
    curSL~setPageStep(10)
    curSL~setTickFrequency(10)
  end
```

Tree View Controls

The tree-view control is a dialog control that displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk. Each item consists of a label and an optional image bitmap, and can have a list of subitems associated with it. By clicking on an item, the user can expand and collapse the associated list of subitems.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with tree-view controls:

Instantiation:

Use the *newTreeView* method of the *dialog* object to retrieve a tree-view object.

Dynamic Definition:

To dynamically create a tree-view in the dialog template of a *UserDialog* use the *createTreeView* method.

Event Notification

To connect the *event* notifications sent by the underlying tree-view control to a method in the Rexx dialog object use the *connectTreeViewEvent* method.

26.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with tree-view objects, including the pertinent methods from other classes:

Table 26.1. TreeView Instance Methods

Method	Description
Useful	**External Methods**
newTreeView	Returns a **`TreeView`** object for the control with the specified ID.
createTreeView	Creates a tree-view control in the dialog template of a *UserDialog*.
connectTreeViewEvent	Connects tree-view event notifications to a method in the Rexx dialog object.
Instance Methods	**Instance Methods**
add	Adds a new item to the tree-view.
child	Child
collapse	Collapse
collapseAndReset	CollapseAndReset
delete	delete
deleteAll	deleteAll
dropHighlight	DropHighlight
dropHighlighted	dropHighlighted
edit	Edit
endEdit	endEdit
ensureVisible	ensureVisible
expand	Expand
find	Finds the first tree-view item whose label matches the text specified. The search is case insensitive.

Method	Description
firstVisible	firstVisible
getImageList	Retrieves the tree-view's image list for the type specified.
getItemData	Gets the item data associated with the specified tree-view item.
getItemHeight	Gets the height the tree-view control used for each of items items.
getItemRect	Retrieves the bounding rectangle for a tree-view item and indicates whether the item is visible.
getStateImage	Returns the index in the image list of the state image for the specified tree-view item.
getToolTips	Retrieves the child *ToolTip* control used by this tree-view.
itemText	Gets the text, the label, of the specified tree-view item.
hitTestInfo	Determines the location of a point relative to the tree-view control.
indent	Indent
indent=	Indent=
insert	Inserts a new item into the tree-view control.
isAncestor	IsAncestor
itemInfo	The *itemInfo* method returns the attributes of the specified item as a stem.
items	items
makeFirstVisible	MakeFirstVisible
modify	Modifies some or all of an item's attributes.
moveItem	moveItem
next	next
nextVisible	NextVisible
parent	Parent
previous	previous
previousVisible	PreviousVisible
removeItemData	Removes and returns the item data associated with the specified tree-view item.
root	Root
select	select
selected	selected
setImageList	Assigns, or removes, an image list for the tree-view control.
setItemData	Sets the item data associated with the specified tree-view item.
setItemHeight	Sets the height of the tree-view items.
setStateImage	Assigns the index in the image list of the state image for the specified tree-view item.
setToolTips	Sets the child *ToolTip* control used by this tree-view.
sortChildren	SortChildren
sortChildrenCB	Has the tree-view sort the children of the specified item by invoking a custom comparsion method in the Rexx dialog.
toggle	Toggle

Method	Description
visibleItems	VisibleItems

26.2. newTreeView (dialog object method)

Tree-view objects can not be instantiated by the programmer from Rexx code. Rather a tree-view object is obtained by using the *newTreeView* method of the *dialog* object. The syntax is:

```
>>--newTreeView(--id--)--------------------------><
```

26.3. createTreeView (UserDialog method)

A tree-view object can be created in the dialog template for a *UserDialog* dialog through the *createTreeView* method. The basic syntax is:

```
>>--createTreeView(-id-,--x-,--y-,--cx-,--cy-+---------+--+------------+--)----><
                                             +-,-style-+  +-,-attrName-+
```

26.4. connectTreeViewEvent (dialog object method)

To connect event notifications from a tree-view object use the *connectTreeViewEvent* method of the *dialog* object. The basic syntax is:

```
>>--connectTreeViewEvent(--id--,--event--+---------------+--)------------------><
                                         +-,--methodName-+
```

26.5. add

```
          +---+
          V   |
>>--add(--+---+-+-------+-+-------+-+----------+-+-------+-+---------+-+-------+--)---><
          +-,-+ +-,-txt-+ +-,-img-+ +-,-selImg-+ +-,-opt-+ +-,-child-+ +-,-usr-+
```

Adds an item with all its attributes to a tree-view control.

Arguments:

The arguments are:

, [optional]

The number of commas specifies at which parent the item is to be inserted. If you omit this argument, the item is inserted as a root item. Each additional comma inserts the item one level deeper than the item inserted *previously*.

txt [optional]

The text for the label of the item. If this arugment is omitted then the empty string is used as the text.

img [optional]

Index in the tree-view control's normal *image* list for the icon image to use when the item is in the nonselected state.

selImg [optional]
: Index in the tree-view control's normal *image* list for the icon image to use when the item is in the selected state.

opt [optional]
: A list of 0 or more of the following keywords separated by spaces, case is not significant. If this argument is omitted the item state will be normal, not expanded:

BOLD	EXPANDED

BOLD
: The item is shown bolded.

EXPANDED
: The item's list of child items is currently expanded, that is, the child items are visible. Only parent items can have this state.

children [optional]
: Used to signal the underlying tree-view control that the item has children, even if no children are inserted. Specify 1 to signal that the item does have children. This is really only useful if the tree-view has the *BUTTONS* style, see the remarks section. When this argument is omitted or less than 1, it is in effect ignored.

usr [optional]
: Associates the specified value with the tree-view item being added. This value is commonly referred to as the *item data* for the item. This can be any Rexx object that the programmer wants to use. The *getItemData* method can be used to retrieve this value. The *setItemData* method, and several other methods, can also be used to set the item data for a tree-view item.

Return value:

Returns the handle of the newly added item on success, or zero on error.

Remarks:

When a tree-view controls has the BUTTONS style, it displays plus and minus buttons next to *parent* items. It does not display the buttons for any item that has no children. However, it is sometimes useful to dynamically insert child items when the button is clicked. Specifying 1 for the *children* argument forces the tree-view control to use a plus button for the item, even if the item has no children.

Associating a item data value with some or all of a tree-view items can be useful in a number of ways. The *itemData* argument allows the programmer to assoicate the value at the time the item in added to the tree-view. The *setItemData* method can be used to assign a data value at any time. The *modify* method can be used to change an item's attributes, including the user data of the item.

Example:

To get the following tree view:

- Peter
  - Mike
    - George
    - Monique
      - John

        - Chris
    - Maud
    - Ringo
- Paul
    - Dave
    - Sam
    - Jeff
- Mary
    - Helen
    - Michelle
    - Diana

The program code should look similar to this:

```
::method initDialog
  treeView = self~newTreeView("IDC_TV_INLAWS")

  treeView~add("Peter", , ,"BOLD EXPANDED")
  treeView~add(,"Mike", , ,"EXPANDED")
  treeView~add(, ,"George")
  treeView~add(, ,"Monique")
  treeView~add(, , ,"John")
  treeView~add(, ,"Chris")
  treeView~add(,"Maud")
  treeView~add(,"Ringo")
  treeView~add("Paul", , ,"BOLD EXPANDED")
  treeView~add(,"Dave")
  treeView~add(,"Sam")
  treeView~add(,"Jeff")
  treeView~add("Mary", , ,"BOLD EXPANDED")
  treeView~add(,"Helen")
  treeView~add(,"Michelle")
  treeView~add(,"Diana")
```

26.6. Child

```
>>--child(--hItem--)----------------------------><
```

The Child method retrieves the first child item of *hItem*.

Arguments:

The only argument is:

hItem

The handle to the item of which the first child is to be retrieved.

Return value:

The handle to the first child item, or -1 if you omitted *hItem*, or 0 in all other cases.

Example:

The following example displays the name of parent of the selected item:

```
::method Child
  curTree = self~newTreeView("IDC_TREE")
  itemInfo. = curTree~itemInfo(curTree~Child(curTree~selected))
  say ItemInfo.!Text
```

26.7. Collapse

```
>>--collapse(--hItem--)--------------------------><
```

The Collapse method collapses the list of child items associated with the specified parent item.

Arguments:

The only argument is:

hItem

The handle to the parent item to collapse.

Return value:

0

The list of child items has collapsed.

-1

hItem is not specified or is 0.

1

For all other cases.

Example:

The following example collapses the selected item:

```
::method CollapseSelected
  curTree = self~newTreeView("IDC_TREE")
  curTree~Collapse(curTree~selected)
```

26.8. CollapseAndReset

```
>>--collapseAndReset(--hItem--)------------------><
```

The CollapseAndReset method collapses the list of child items associated with the specified parent item and removes the child items.

Arguments:

The only argument is:

hItem

The handle to the parent item to collapse.

Return value:

0

The list of child items has collapsed and the child items have been removed.

-1
: *hItem* is not specified or is 0.

1
: For all other cases.

Example:

The following example collapses the selected item and removes all its child items:

```
::method CollapseSelectedAndReset
  curTree = self~newTreeView("IDC_TREE")
  curTree~CollapseAndReset(curTree~selected)
```

26.9. delete

```
>>--delete(--hItem--)----------------------------><
```

The delete method removes an item from a tree-view control.

Arguments:

The only argument is:

hItem
: The handle to the item to be deleted.

Return value:

0
: The item was deleted.

1
: An error occurred.

-1
: *hItem* is 0 or is not a valid value.

Example:

The following example deletes the selected item and all its children, if any:

```
::method IDC_PB_DELETE
  curTree = self~newTreeView("IDC_TREE")
  curTree~delete(curTree~selected)
```

26.10. deleteAll

```
>>--deleteAll------------------------------------><
```

The deleteAll method removes all items from a tree view control.

Return value:

0
: The items were removed.

1
: For all other cases.

26.11. DropHighlight

```
>>--dropHighlight(--hItem--)---------------------><
```

The DropHighlight method redraws *hItem* in the style used to indicate the target of a drag-and-drop operation.

Arguments:

The only argument is:

hItem
: The handle of the item to be redrawn.

Return value:

0
: The item was redrawn.

-1
: *hItem* was not specified or is 0.

1
: For all other cases.

26.12. dropHighlighted

```
>>--dropHighlighted------------------------------><
```

The dropHighlighted method retrieves the item that is the target of a drag-and-drop operation.

Return value:

The handle to the item, or 0 in all other cases.

26.13. edit

```
>>--edit(--hItem--)------------------------------><
```

The *edit* method starts editing the text of the specified item by replacing the text with a single-line edit control containing this text. It implicitly selects and focuses the specified item.

Arguments:

The only argument is:

hItem
: The handle to the item to be edited.

Return value:

On success, the handle to the edit control used to edit the item text. On error, -1 if *hItem* is 0, or 0 for all other errors.

26.14. endEdit

```
>>--endEdit(--+--------+--)----------------------><
              +-cancel-+
```

The endEdit method ends the editing of the item label of the tree view.

Arguments:

The only argument is:

cancel

Indicates whether editing is canceled without being saved to the label. If you specify "1" or "YES", editing is canceled. Otherwise, the changes are saved to the label, which is the default.

Return value:

0

Editing has ended successfully.

1

For all other cases.

26.15. ensureVisible

```
>>--ensureVisible(--hItem--)---------------------><
```

The ensureVisible method ensures that a tree-view item is visible, expanding the parent item or scrolling the tree-view control, if necessary.

Arguments:

The only argument is:

hItem

The handle to the item to be visible.

Return value:

0

The items in the tree-view control were scrolled to ensure that the specified item is visible.

-1

hItem is not specified or is 0.

1

For all other cases.

26.16. Expand

```
>>--expand(--hItem--)----------------------------><
```

The Expand method expands the list of child items associated with the specified parent item.

Arguments:

The only argument is:

hItem
: The handle to the parent item to be expanded.

Return value:

0
: The parent item was expanded.

-1
: *hItem* is not specified or is 0.

1
: For all other cases.

Example:

The following example expands the selected item:

```
::method ExpandSelected
  curTree = self~newTreeView("IDC_TREE")
  curTree~Expand(curTree~selected)
```

26.17. find

```
>>--find(--text--+-------------+--+----------+--)---------------><
                 +-,-startItem-+  +-,-abbrev-+
```

Finds the first tree-view item whose label matches the text specified. The search is case insensitive.

Arguments:

The arguments are:

text [required]
: Specifies the label, the text, of item to search for. If the *abbrev* arugment is omitted or false, the search is case insensitive, but is otherwise exact. I.e., *Coffee Product* will not match *Coffee Products*, but will match *coffee product* or *coffee PRODUCT*

startItem [optiona]
: The tree-view item to begin the search at. If omitted, the search begins at the root of the tree.

abbrev [optiona]
: True to match any item whose label begins with *text*. False to exactly match *text*. The default is false.

  If *abbrev* is true and *text* is *Coffee product*, then the labels *coffee product*, *COFFEE products*, and *coffee production* will all match.

Return value:

Returns the *handle* of the first item found that matches the search specifications, or 0 if no item is found.

Remarks:

The search starts at the tree-view item specified, the root item by default, and recursively searches all children and grandchildren of the start point. There is only one root item in a tree-view control. Microsoft defines the root in a tree-view to be the first item added. It is possible, but unusual, to add a *parent* tree-view item at the same level as the root item that is not a child of the root. It is important to note that the search is not a search of *every* item in the tree-view, it is a search of every child and grandchild of the start point.

Example:

This example searches for the first tree-view item whose label is *Computers*, and if found, selects that item.

```
  tv = self~newTreeView(IDC_TV_PRODUCTS)

  hItem = tv~find('Computers')
  if hItem \== 0 then tv~select(hItem)
```

26.18. firstVisible

```
>>--firstVisible---------------------------------><
```

The firstVisible method retrieves the first visible item in the client window of a tree-view control.

Return value:

The handle to the first visible item, or 0 in all other cases.

Example:

The following example displays the name of the first visible item:

```
::method FirstVisibleName
  curTree = self~newTreeView("IDC_TREE")
  itemInfo. = curTree~itemInfo(curTree~firstVisible)
  say ItemInfo.!Text
```

26.19. getImageList

```
>>--getImageList(--+--------+--)-----------------><
                   +--type--+
```

Retrieves the current image list for the type specified. The default is to retrieve the tree-view's normal image list. See the *setImageList* method for infomation on assigning an image list to a tree-view control.

Arguments:

The single optional argument is:

type [optional]

Specifies which image list, the NORMAL or STATE image list, to retrieve the image list for. The operating system uses these 2 flags to specify the 2 types of image lists.

- TVSIL_NORMAL: Numeric value == 0. Use the NORMAL keyword or the numeric value. The *toID* method of the *Image* class can also be used with TVSIL_NORMAL to get the correct numeric value.
- TVSIL_STATE: Numeric value == 2. Use the STATE keyword or the numeric value. The *toID* method of the *Image* class can also be used with TVSIL_STATE to get the correct numeric value.

If this argument is omitted, the default is TVSIL_NORMAL.

Return value:

This method returns the current image *ImageList*, if there is one. The **`.nil`** object is returned if there is no current image list of the type specified.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example releases the tree-view's image lists, if they exist, when the dialog ends.

```
::method ok
  self~doImageLists return self~ok:super

::method cancel
  self~doImageLists
  return self~cancel:super

::method doImageLists
  expose treeView

  types = .array~of('NORMAL', 'STATE')

  do type over types
    il = treeView~getImageList(type)
    if il \== .nil then il~release
  end
```

26.20. getItemData

```
>>--getItemData(--hItem--)-----------------------><
```

Gets the item data associated with the specified tree-view item.

Arguments:

The single argument is:

hItem [required]

The tree-view item to retrieve the item data from.

Return value:

Returns the item data associated with the specified item, or the **`.nil`** object if there is no data associated with the item.

Remarks:

The tree-view control allows the user (the Rexx programmer) to associate a data value with any, or all, of the tree-view items. The data value can be any ooRexx object. The data value is set using the *setItemData* method. If needed the data value can be removed from a tree-view item through the *removeItemData* method. Storing a user value for each item can be useful in any number of ways.

Details

Raises syntax errors when incorrect usage is detected.

26.21. getItemHeight

```
>>--getItemHeight---------------------------------------------><
```

Returns the current height of the tree-view items.

Arguments:

There are no arguments to this method.

Return value:

Returns the current height of the tree-view items in pixels.

Remarks:

The tree-view uses the value returned for the height of all items. The value itself is the height of a single item. See the *setItemHeight* method.

26.22. getItemRect

```
>>--getItemRect(--hItem--,--rect--+-------------+--)------------><
                                  +-,-textOnly--+
```

Retrieves the bounding rectangle for a tree-view item and indicates whether the item is visible.

Arguments:

The arguments are:

hItem [required]

The tree-view item to retrieve the bounding rectangle for.

rect [required in / out]

A *Rect* object used to return the bounding rectangle coordinates. Coordinates are in pixels expressed as *client* area coordinates.

textOnly [optional]

If *textOnly* is true, the bounding rectangle returned is for the text portion only of the item. If false, the rectangle includes the entire line the item occupies in the tree-view control. If the argument is omitted, the default is true.

Return value:

Returns true if the item specified is visible and the bounding rectangle is filled in. Returns false if the item is not visible and the bounding rectangle is not filled in.

Remarks:

The coordinates of the bounding rectangle are only filled in if the item is visible.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows the event handler for a tool tip used for a tree-view. When the tool tip is about to be shown, the bounding rectangle for the text only portion of the item is used to reposition the tool tip so that it is in line with the item and to the right of the item:

```
::method onShow unguarded
  expose currentItem
  use arg toolTip, treeView

  pos = toolTip~getRealPos

  r = .Rect~new
  if treeView~getItemRect(currentItem, r) then do
    treeView~client2screen(r)

    pos~x = r~right + 5
    pos~y = r~top
    toolTip~moveTo(pos)
    return .true
  end

  return .false
```

26.23. getStateImage

```
>>--getStateImage(--hItem--)---------------------><
```

Returns the index in the image list of the state image for the specified tree-view item.

Arguments:

The single argument is:

hItem [required]

The handle of the tree-view item whose state image index is being queried.

Return value:

Returns -1 on error, otherwise returns the index for the state image. A value of 0 is used by the operating system to indicate there is no state image assigned for a tree-view item.

Remarks:

State images ares displayed next to an item's icon, immediately to the left. They are used to indicate an application defined state for the item. The state image list is set using the *setImageList* method. A value of 0 indicates there is no state image. Only 15 state images can be assigned, indexes 1 through 15.

Details

Raises syntax errors when incorrect usage is detected.

26.24. getToolTips

```
>>--getToolTips----------------------------------><
```

Retrieves the child *ToolTip* control used by this tree-view.

Arguments:

There are no arguments for this method

Return value:

Returns the Rexx *ToolTip* object that represents the ToolTip control of the tree-view, or the **`.nil`** object if the tree-view does not currently have a ToolTip.

26.25. hitTestInfo

```
Form 1:

>>--hitTestInfo(--pt--)--------------------------><

Form 2:

>>--hitTestInfo(--x,--y--)-----------------------><


Generic form:

>>--hitTestInfo(--pointToTest--)-----------------><
```

Determines the location of a point relative to the *client area* of the tree-view control.

Arguments:

The argument(s) specify the coordinates of the point to test. These coordinates can be specified using either a *Point* object, or by the x and y coordinate of the point:

pointToTest [required]

The x, y coordinates of the point to test. As noted either a **`Point`** object or the whole number x and y coordinates can be used to specify the point. If a **`Point`** is not used, both *x* and *y* are required.

Return value:

The return is a **`Directory`** object whose indexes contain the result of the hit test. On return, the **`Directory`** object will have the following indexes with the corresponding information:

hItem

The *handle* of the tree-view item under the point specified, if there is one. If the point does not hit an item, the value will be 0.

location

A string of blank separated keywords describing the location of the point. For instance, the string might be "ONITEM ONLABEL" if the point hits an item, on the item's label. Or it could be "ABOVE TORIGHT" if the point is not on the client area of the tree-view at all. The possible descriptive words are as follows.

ABOVE	ONBUTTON	ONRIGHT
BELOW	ONICON	ONSTATEICON
NOWHERE	ONINDENT	TOLEFT
ONITEM	ONLABEL	TORIGHT

ABOVE

Above the client area.

BELOW

Below the client area

NOWHERE
: In the client area, but below the last item.

ONITEM
: On the bitmap or label associated with an item.

ONBUTTON
: On the button associated with an item.

ONICON
: On the bitmap associated with an item.

ONINDENT
: In the indentation associated with an item.

ONLABEL
: On the label (text) associated with an item.

ONRIGHT
: In the area to the right of an item.

ONSTATEICON
: On the state icon for a tree-view item that is in a user-defined state.

TOLEFT
: To the left of the client area.

TORIGHT
: To the right of the client area.

Remarks:

The tree-view will reply for any given point, even invalid points like (-7000, -6000). Points not on the tree-view control itself will have the **`hItem`** index set to 0 and the **`location`** string will show where the point is in relation to the control. For example (-7000, -6000) will have a location string of "ABOVE TOLEFT".

Details:

Raises syntax errors when incorrect arguments are detected.

26.26. Indent

```
>>--indent---------------------------------------><
```

The Indent method retrieves the amount, in pixels, by which the child items are indented relative to their parent item.

Return value:

The amount indented, in pixels.

26.27. Indent=

```
>>--indent=indent-----------------------------------><
```

The Indent= method sets the width of indentation for a tree-view control and redraws the control to reflect the new width.

Arguments:

The only argument is:

indent

The width of the indentation, in pixels. If you specify a width that is smaller than the system-defined minimum, it is set to the system-defined minimum.

Return value:

-1 if *indent* is 0.

26.28. insert

```
>>--insert(--+----------+-+---------+-+-------+-+-------+-+----------+--------->
             +--parent--+ +-,-after-+ +-,-txt-+ +-,-img-+ +-,-selImg-+

>------------+---------+-+------------+-+------------+-------------------------><
             +-,-state-+ +-,-children-+ +-,-itemData-+
```

Inserts a new item into the tree-view control.

Arguments:

The arguments are:

parent [optional]

Handle of the parent to the new item. If this argument is omitted, or is the keyword *root*, case insignificant, then the new item is inserted a the root of the tree-view.

after [optional]

Either a tree-view item handler, or one of the following keywords separated by spaces, case is not significant. This specifies where the new item will be inserted. When *after* is an item handle, the new item is inserted after the specified item. Otherwise the item is inserted as explained for each keyword:

FIRST	LAST	SORT

FIRST

Inserts the item at the beginning of the list.

LAST

Inserts the item at the end of the list. This is the default if the *after* argument is omitted

SORT

Inserts the item into the list in alphabetical order.

txt [optional]

The text for the label of the item. If this arugment is omitted then the empty string is used as the text.

img [optional]
Index in the tree-view control's normal *image* list of the icon image to use when the item is in the nonselected state.

selImg [optional]
Index in the tree-view control's normal *image* list of the icon image to use when the item is in the selected state.

state [optional]
A list of 0 or more of the following keywords separated by spaces, case is not significant. If this argument is omitted the item state will be normal, not expanded:

BOLD	EXPANDED

BOLD
The item is shown bolded.

EXPANDED
The item's list of child items is currently expanded, that is, the child items are visible. Only parent items can have this state.

children [optional]
Used to signal the underlying tree-view control that the item has children, even if no children are inserted. Specify 1 to signal that the item does have children. This is really only useful if the tree-view has the *BUTTONS* style, see the remarks section. When this argument is omitted or less than 1, it is in effect ignored.

itemData [optional]
Associates the specified value with the tree-view item being inserted. This can be any Rexx object that the programmer wants to use. The *getItemData* method can be used to retrieve this value.

Return value:

Returns the handle of the newly inserted item on success, or zero on error.

Remarks:

When a tree-view controls has the BUTTONS style, it displays plus and minus buttons next to *parent* items. It does not display the buttons for any item that has no children. However, it is sometimes useful to dynamically insert child items when the button is clicked. Specifying 1 for the *children* argument forces the tree-view control to use a plus button for the item, even if the item has no children.

Associating a item data value with some or all of a tree-view items can be useful in a number of ways. The *itemData* argument allows the programme to assoicate the value at the time the item in added to the tree-view. The *setItemData* method can be used to assign a data value at any time.

Example:

This example inserts a new folder tree-view item as a sibling of the currently selected item. It also assigns both a selected and unselected icon to the item from the tree-view's image list by specifying the index in the image list for the icons:

```
::constant UNSELECTED_FOLDER  0         -- Index for the icon of an unselected folder
 item
::constant SELECTED_FOLDER    1         -- Index for the icon of a selected folder item
::constant UNSELECTED_LEAF    2         -- Index for the icon of an unselected leaf item
::constant SELECTED_LEAF      3         -- Index for the icon of a selected leaf item
```

```
...

treeControl~insert(treeControl~Parent(selected), , text, self~UNSELECTED_FOLDER,
 self~SELECTED_FOLDER)
```

26.29. IsAncestor

```
>>--isAncestor(--hParent--,--hItem--)------------><
```

The IsAncestor method checks if an item is an ancestor of another item.

Arguments:

The arguments are:

hParent

The ancestor.

hItem

The item to be checked.

Return value:

1 if *hParent* is an ancestor of *hItem*.

26.30. itemInfo

```
>>--itemInfo(--hItem--)-------------------------><
```

The *itemInfo* method returns the attributes of the specified item as a stem.

Arguments:

The arguments are:

hItem [required]

The handle of the tree-view item whose attributes are to be returned.

Return value:

Returns a stem on success, or -1 if an error occurred. The tails, (the indexes,) of the stem will be:

!TEXT

The text, the label, of the item.

!CHILDREN

Whether the item has children or not. True if the item has children, false if it does not.

!IMAGE

The index in the image list of the normal icon for the item. The index will be 0 if there is no index assigned to the item. However, 0 is also a valid index in a the image list. There is no way to distinguish the two.

!SELECTEDIMAGE

The index in the image list of the selected icon for the item. The index will be 0 if there is no index assigned to the item. However, 0 is also a valid index in a the image list. There is no way to distinguish the two.

!STATE

A string consisting of 0 or more of the following keywords:

EXPANDED

The item's list is currently expanded with all child items visible. This only applies to parent items.

BOLD

The item is in bold.

SELECTED

The item is selected.

EXPANDEDONCE

The item's list has been expanded at least once. This only applies to parent items.

INDROP

The item is selected as a drag-and-drop target.

CUT

The item is selected as part of a cut-and-paste operation.

EXPANDPARTIAL

Indicates a partially expanded tree-view item. In this state, some, but not all, of the child items are visible and the parent item's plus symbol is displayed.

!ITEMDATA

The item data value, if a value has been assigned to the item. If no item data value has been assigned, the `.nil` object is returned.

Remarks:

The item data value can be assigned to a tree-view item through a number of methods, such as the *setItemData*, *insert*, or *modify* methods.

Example:

The following example displays the attributes of the selected item:

```
::method Info
  tree = self~newTreeView(IDC_TREE)
  itemInfo. = curTree~itemInfo(tree~selected)
  say itemInfo.!TEXT
  say itemInfo.!CHILDREN
  say itemInfo.!IMAGE
  say itemInfo.!STATE
  say itemInfo.!ITEMDATA
```

26.31. items

```
>>--items----------------------------------------><
```

The items method retrieves the number of items in a tree-view control.

Return value:

The number of items.

Example:

The following example counts all items in a tree-view control:

```
::method Count
  curTree = self~newTreeView("IDC_TREE")
  say curTree~items
```

26.32. itemText

```
>>--itemText(--hItem--)--------------------------><
```

Retrieves the text of the specified tree-view item.

Arguments:

The single argument is:

hItem [required]

The tree-view item to retrieve the text from.

Return value:

The return is the text assigned to the specified tree-view item.

Example:

This example retrieves the text of the selected item:

```
  tv = self~newTreeVies(IDC_TV_BOOK)

  selectedItem = tv~selected
  if selectedItem == 0 then do
      say 'No item is selected'
  end
  else do
      say 'The text for the selected item is:' tv~itemText(selectedItem)
  end
```

26.33. MakeFirstVisible

```
>>--makeFirstVisible(--hItem--)------------------><
```

The MakeFirstVisible method ensures that *hItem* is visible and displays it at the top of the control's dialog, if possible. If the specified item is near the end of the control's hierarchy of items, it might not become the first visible item depending on how many items fit in the dialog.

Arguments:

The only argument is:

hItem

The handle to the item to be visible first.

Return value:

0

The item is visible first.

-1
: *hItem* was not specified or is 0.

1
: For all other cases.

26.34. modify

```
>>--modify(--+---------+-+-------+-+-------+-+----------+---------------------->
             +--hItem--+ +-,-txt-+ +-,-img-+ +-,-selImg-+

>------------+---------+-+------------+-+------------+------------------------><
             +-,-state-+ +-,-children-+ +-,-itemData-+
```

Modifies some or all of an item's attributes.

Arguments:

The arguments are:

hItem [optional]
: Handle of the item to modify. If this argument is omitted the currently selected item is used. If the item is omitted and there is no currently selected item, -1 is returned as an error.

txt [optional]
: The text for the label of the item. If this argument is omitted then the text is not changed.

img [optional]
: Index in the tree-view control's normal *image* list of the icon image to use when the item is in the nonselected state. If omitted, then the index is not changed.

selImg [optional]
: Index in the tree-view control's normal *image* list of the icon image to use when the item is in the selected state. If omitted the selected icon index is not changed.

state [optional]
: Modifies the item state. Use 0 or more of the following keywords separated by spaces, case is not significant. If this argument is omitted, or the empty string, the item state is not changed.

BOLD	NOTDROP	EXPANDEDONCE
NOBOLDBOLD	SELECTED	NOTEXPANDEDONCE
EXPANDED	NOTSELECTED	EXPANDPARTIAL
NOTEXPANDED	CUT	NOTEXPANDPARTIAL
DROP	NOTCUT	

BOLD
: The item is shown bolded.

NOTBOLD
: The item is shown not bolded.

EXPANDED
: The item's list of child items is currently expanded, that is, the child items are visible. Only parent items can have this state.

NOTEXPANDED
: The item's list is currently not expanded.

DROP
: The item is selected as a drag-and-drop target.

NOTDROP
: The item is not selected as a drag-and-drop target.

SELECTED
: The item is selected. Its appearance depends on whether it has the focus and whether the system colors are used for the selection.

NOTSELECTED
: The item is not selected.

CUT
: The item is selected as part of a cut-and-paste operation.

NOTCUT
: The item is not selected as part of a cut-and-paste operation.

EXPANDEDONCE
: The item's list of child items has been expanded at least once.

NOTEXPANDEDONCE
: The item's list of child items has not been expanded at least once.

EXPANDPARTIAL
: A partially expanded tree-view item. In this state, some, but not all, of the child items are visible and the parent item's plus symbol is displayed.

NOTEXPANDPARTIAL
: The item is not in a partially expanded state.

children [optional]
: Used to signal the underlying tree-view control that the item has children, even if no children are inserted. Specify 1 to signal that the item does have children. This is really only useful if the tree-view has the *BUTTONS* style. When this argument is omitted the item's *children* value is not changed.

itemData [optional]
: Associates the specified value with the tree-view item being inserted. This can be any Rexx object that the programmer wants to use. The *getItemData* method can be used to retrieve this value. If this argument is omitted, the item's item data value is not changed.

Return value:

Returns one of the following values:

0
: The item has been modified.

  The tree-view control reported an error when modifying the item.

-1
: The *hItem* argument was not valid.

Example:

The following example changes the text of the item to bold when it is selected:

```
::method onSelectChanging
  tree = self~newTreeView(IDC_TREE)
  tree~modify(tree~selected, , , ,"BOLD")
```

26.35. moveItem

```
>>--moveItem(--hItem--,--hNewParent--,--redraw--+---------------+--)-----------><
                                                +-,--"NODELETE"-+
                                                +-,--"SIBLINGS"-+
```

The moveItem method moves an item to a new location.

Arguments:

The arguments are:

hItem

The handle to the item to be moved.

hNewParent

The handle to the new parent to which the item is to be moved.

redraw

The tree-view control is updated.

options

One of the following options:

"NODELETE"

The item is copied to another location.

"SIBLINGS"

Siblings are moved together with the item.

Return value:

The handle to the new parent, or 0 in all other cases.

26.36. next

```
>>--next(--hItem--)------------------------------><
```

The next method retrieves the sibling item next to *hItem*.

Arguments:

The only argument is:

hItem

The handle to the item next to the sibling item to be retrieved.

Return value:

The handle to the next sibling item, or -1 if you omitted *hItem*, or 0 in all other cases.

Example:

The following example displays the name of the selected item and its siblings:

```
::method SiblingNames
  curTree = self~newTreeView("IDC_TREE")
  nextItem = curTree~selected
  do while nextItem \= 0
    itemInfo. = curTree~itemInfo(nextItem)
    say ItemInfo.!Text
    nextItem = curTree~next(nextItem)
  end
```

26.37. NextVisible

```
>>--nextVisible(--hItem--)-----------------------><
```

The NextVisible method retrieves the visible item following *hItem*.

Arguments:

The only argument is:

hItem

The handle to the item that precedes the visible item to be retrieved. *hItem* must also be visible.

Return value:

The handle to the next visible item, or -1 if you omitted *hItem*, or 0 in all other cases.

26.38. Parent

```
>>--parent(--hItem--)----------------------------><
```

The Parent method retrieves the parent of the specified item.

Arguments:

The only argument is:

hItem

The handle to the item for which the parent is to be retrieved.

Return value:

The handle to the parent item, or -1 if *hItem* is not specified or is 0, or 0 in all other cases.

Example:

The following example displays the name of the selected item's parent:

```
::method Parent
  curTree = self~newTreeView("IDC_TREE")
  itemInfo. = curTree~itemInfo(curTree~Parent(curTree~selected))
  say ItemInfo.!Text
```

26.39. previous

```
>>--previous(--hItem--)--------------------------><
```

The previous method retrieves the sibling item preceding *hItem*.

Arguments:

The only argument is:

hItem

The handle to the item that follows the sibling item to be retrieved.

Return value:

The handle to the previous sibling item, or -1 if *hItem* is not specified or is 0, or 0 in all other cases.

26.40. PreviousVisible

```
>>--previousVisible(--hItem--)-------------------><
```

The PreviousVisible method retrieves the visible item preceding *hItem*.

Arguments:

The only argument is:

hItem

The handle to the item that follows the visible child item to be retrieved. *hItem* must also be visible.

Return value:

The handle to the previous visible child item, or -1 if *hItem* is not specified or is 0, or 0 in all other cases.

26.41. removeItemData

```
>>--removeItemData(--hItem--)--------------------><
```

Removes and returns the item data associated with the specified tree-view item.

Arguments:

The single argument is:

hItem [required]

The tree-view item whose data value is being removed.

Return value:

Returns the item data value associated with the specified item, or `.nil` if there is no data associated with the item.

Remarks:

The tree-view control allows the user (the Rexx programmer) to associate a data value with any, or all, of the tree-view items. The data value can be any ooRexx object. The data value is set using the *setItemData* method. The data value can be retrieved without removing it from a tree-view item through the *getItemData* method. Storing a user value for each item can be useful in any number of ways.

Details

Raises syntax errors when incorrect usage is detected.

26.42. root

```
>>--root-----------------------------------------><
```

The *root* method retrieves item handle of the first or topmost item of the tree-view control.

Arguments:

This method has not arguments.

Return value:

The handle to the first or topmost item on success. 0 on error.

Example:

The following example displays the text, or label, of the root item:

```
::method rootLabel
  tv = self~newTreeView(IDC_TV_ORG_CHART)
  say 'The head of the organization is' tv~getItemText(tv~root)
```

26.43. select

```
>>--select(--hItem--)----------------------------><
```

The select method selects a specific item.

Arguments:

The only argument is:

hItem

The handle to the item to be selected.

Return value:

0

The item was selected.

-1

hItem was not specified or is 0.

1

For all other cases.

26.44. selected

```
>>--selected-------------------------------------><
```

The selected method retrieves the currently selected item.

Return value:

The handle to the currently selected item, or 0 in all other cases.

Example:

The following example displays the name of the selected item:

```
::method SelectedName
  curTree = self~newTreeView("IDC_TREE")
  itemInfo. = curTree~itemInfo(curTree~selected)
  say ItemInfo.!Text
```

26.45. setImageList

```
>>--setImageList(--+----------------+--+----------+--)---------><
                   +--newImageList--+  +--,--type--+
```

Assigns, or removes, an image list for the tree-view control.

Arguments:

The arguments are:

newImageList [optional]

The *ImageList* object to assign to the tree-view. If this argument is omitted or is the **.nil** object, the existing image list, if any, is removed.

type [optional]

Specifies which image list, the NORMAL or STATE image list, to assign or remove. The operating system uses these 2 flags to specify the 2 types of image lists.

- TVSIL_NORMAL: Numeric value == 0. Use the NORMAL keyword or the numeric value. The *toID* method of the *Image* class can also be used with TVSIL_NORMAL to get the correct numeric value.
- TVSIL_STATE: Numeric value == 2. Use the STATE keyword or the numeric value. The *toID* method of the *Image* class can also be used with TVSIL_STATE to get the correct numeric value.

If this argument is omitted, the default is TVSIL_NORMAL.

Return value:

The existing image list is returned, if there is one. Otherwise, **.nil** is returned.

Remarks:

Tree-view controls can have two image lists. The normal image list, which contains selected, nonselected, and overlay images for the items of a tree-view control. And, the state image list. The state images can be used by the programmer to indicate application-defined item states. A state image is displayed to the left of an item's selected or nonselected image.

The tree-view control does not destroy an image list that is associated with it. The programmer must destroy the image list separately, if desired. This is useful if the same image list is assigned to multiple tree-view controls. In essence, the ownership of the image list remains with the programmer. The *ImageList* and *Image* classes are used to manage image lists and images in ooDialog. The documentation on both classes discusses when and why the programmer may want to release image lists. The Image class documentation has the most detail on this subject.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example creates an image list and assigns it to the tree-view control for use as the normal image list. Since the normal is the default, the *type* argument does not need to be specified:

```
/* set images for the items */
image = .Image~getImage("bmp\psdemotv.bmp")
imageList = .ImageList~create(.Size~new(32, 32), .Image~toID(ILC_COLOR8), 10, 0)
if \image~isNull,  \imageList~isNull then do
   imageList~add(image)
   tc~setImageList(imageList)
   image~release
end
```

26.46. setItemData

```
>>--setItemData(--hItem,--+--------+--)----------><
                          +-,-data-+
```

Sets the item data associated with the specified tree-view item.

Arguments:

The arguments are:

hItem [required]

The tree-view item the data value is associated with.

data [optional]

The data value to store for the tree-view item. This can be any Rexx object. (Recall that in ooRexx numbers and strings are also objects.) If this argument is omitted, the current item data, if any, is removed.

Return value:

Returns the previous item data associated with the tree-view item, or **`.nil`** if there was no previous data. On error, **`.nil`** is returned and the *.systemErrorCode* is set. An error is highly unlikely.

Remarks:

The tree-view control allows the user (the Rexx programmer) to associate a data value with any, or all, of the tree-view items. The data value can be any ooRexx object. The data value can be retrieved without removing it from a tree-view item through the *getItemData* method, or it can be removed through the *removeItemData*. Storing a user value for each item can be useful in any number of ways. One specific use is in the *sortChildrenCB* method.

It is possible, although very unlikely, that there will be an error in the tree-view control when setting the item data. In this case **`.nil`** is returned and the **`.systemErrorCode`** is set to 156 ERROR_SIGNAL_REFUSED *The recipient process has refused the signal*.

This is not a system error, the code is just used here to indicate a tree-view error when setting the item data. The tree-view provides no information on why it failed.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

26.47. setItemHeight

```
>>--setItemHeight(--cyItem--)--------------------><
```

Sets the height of the tree-view items.

Arguments:

The single argument is:

cyItem [required]

The new height for every item in the tree view, in pixels. Heights less than 1 will be set to 1. If this argument is not even, it will be rounded down to the nearest even value. If this argument is -1, the control will revert to using its default item height.

Return value:

Returns the previous item height, in pixels.

Remarks:

The tree-view uses the value for the height of all items. The value itself is the height of a single item. See the *getItemHeight* method. By default, the tree-view only allows even heights. However, if the *NONEVENHEIGHT* style is give to the tree-view, both odd and even values can be assigned.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example sets the height of each tree-view item to 20 pixels. (For this specific application the previouse height was 16 pixels.)

```
tv = self~newTreeView(IDC_TREE)
tv~setItemHeight(20)
```

26.48. setStateImage

```
>>--setStateImage(--hItem--,--index--)-----------><
```

Assigns the index in the *image* list of the state image for the specified tree-view item.

Arguments:

The arguments are:

hItem [required]

The handle of the tree-view item whose state image index is being set.

index [required]

The index of the image in the state image list for the specified tree-view item.

Return value:

Returns true on success, false on error.

Remarks:

State images ares displayed next to an item's icon, immediately to the left. They are used to indicate an application defined state for the item. The state image list is set using the *setImageList* method. A value of 0 indicates there is no state image. Only 15 state images can be assigned, indexes 1 through 15.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example ...

```
::method setStateIndex private
    expose tv
    use strict arg hItem, state

    select
        when state == 'singleton' then index = 1
        when state == 'duplicate' then index = 2
        when state == 'triplicate' then index = 3
        when state == 'quadruplicate' then index = 4
        otherwise index = 0
    end

    tv~setStateImage(hItem, index)

    return 0
```

26.49. setToolTips

```
>>--setToolTips(--toolTip--)---------------------><
```

Sets the child *ToolTip* control used by this tree-view.

Arguments:

The arguments are:

toolTip [required]

The Rexx **`ToolTip`** object that represents the tool tip control the tree-view should use.

Return value:

Returns the previous tool tip, as a Rexx **`ToolTip`** object, or the **`.nil`** object if there is no previous tool tip.

Details

Raises syntax errors when incorrect usage is detected.

26.50. SortChildren

```
>>--sortChildren(--hItem--)----------------------><
```

The SortChildren method sorts the child items of the specified parent item in a tree-view control.

Arguments:

The only argument is:

hItem

The handle to the parent item the child items of which are to be sorted.

Return value:

0

The child items were sorted.

-1

hItem was not specified or is 0.

1

For all other cases.

26.51. sortChildrenCB

```
>>--sortChildrenCB(--+---------+--+-----------+--+---------+--)----------------><
                     +--hItem--+  +-,-mthName-+  +-,-param-+
```

The *sortChildrenCB* method causes the tree-view to sort the children of the specified parent tree-view item by invoking a comparison method in the Rexx dialog. The Rexx comparsion method determines the order of the children items.

Arguments:

The arguments are:

hItem [optional]

The parent item of the children to be sorted. If this argument is omitted, the *root* item is used as the parent.

mthName [optional]

The name of the comparsion method in the Rexx dialog to be invoked to determine the relative order of two tree-view items. If this argument is omitted, a default method name of *onSortChildrenCB* is used.

param [optional]

The *param* argument can be any object the programmer wants to use. If the argument exists, it is passed as the third argument to the *mthName* comparison method. If the argument is omitted, the **`.nil`** object is passed as the third argument.

Return value:

Returns true on success, false on error.

Remarks:

The tree-view control allows the user (the Rexx programmer) to associate a user *data* value with any, or all, of the tree-view items. The data value can be any ooRexx object. For the *sortChildrenCB* method to work correctly, the programmer must associate a data value with each tree-view item.

The onSortChildrenCB method

As the tree-view control needs to compare two items it will invoke, through the ooDialog framework, the comparison method in the Rexx dialog named by the *mthName* argument. The return from the Rexx method determines how the items are sorted.

The comparison method must return a whole number and this value is passed on to the tree-view control. The method must return a negative value if the first item should come before the second item, a positive number if the first item should come after the second item, and 0 if the items are equivalent.

The arguments passed to the comparsion method are the item data items associated with the two items being compared. This is why the sort will not work correctly if tree-view items are not assigned a item data item. The comparison method should be coded as follows:

```
::method onSortChildrenCB unguarded
  use arg data1, data2, param
```

Arguments:

The comparison method receives 3 arguments:

data1
: The user *data value* assigned to the first tree-view item to be compared. This data value is the value the programmer assigned to the item using one of the *setItemData*, *insert*, *add*, etc., methods.

data2
: The user *data value* assigned to the second tree-view item to be compared.

param
: The *param* argument passed to the *sortChildrenCB* method, if it exists. If it was omitted, the param argument will be the **`.nil`** object.

Return:

The return must be a whole number. Less than 0 places the first item before the second item, greater than 0 places the first item after the second. Return 0 if the two items are equivalent. The programmer determines what number to return based on the 2 item data values passed to the method.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example sorts the children of the selected item in descending alphabetic order:

```
::method onSortChildren unguarded
    expose tv

    selectedItem = tv~selected
    if selectedItem == 0 then selectedItem = tv~root

    ret = tv~sortChildrenCB(selectedItem, rexxSort)


::method rexxSort unguarded
    use arg itemData1, itemData2, userParam

    -- Reverse sort:
```

```
    return itemData2~compareTo(itemData1)
```

26.52. Toggle

```
>>--toggle(--hItem--)---------------------------><
```

The Toggle method collapses the list of the specified item if it was expanded, or expands it if it was collapsed.

Arguments:

The only argument is:

hItem

The handle to the item to be expanded or collapsed.

Return value:

0

The item was expanded or collapsed.

-1

hItem is not specified or is 0.

1

For all other cases.

Example:

The following example toggles between expanding and collapsing a selected item:

```
::method ToggleSelected
  curTree = self~newTreeView("IDC_TREE")
  curTree~Toggle(curTree~selected)
```

26.53. VisibleItems

```
>>--visibleItems--------------------------------><
```

The VisibleItems method obtains the number of items that can be fully visible in a tree-view control. This number can be greater than the number of items in the control. The control calculates this value by dividing the height of the client window by the height of an item.

Return value:

The number of items that can be fully visible. For example, if you can see all of 19 items and part of another item, the return value is 19, not 20.

Example:

The following example returns the number of items that can be fully visible:

```
::method Visible
  curTree = self~newTreeView("IDC_TREE")
  say curTree~VisibleItems
```

26.54. TvCustomDrawSimple Class

A **`TvCustomDrawSimple`** object is used when a *TreeView* control is registered for simple *custom* draw. A **`TvCustomDrawSimple`** object is passed to the CUSTOMDRAW event handler. It supplies both information to the event handler and returns information back to the Windows control. The attributes of the object convey the information both ways.

When the ooDialog framework receives information from the Windows custom draw control, it assigns values to the attributes of the **`TvCustomDrawSimple`** object and sends the object to the Rexx dialog's event handler. The programmer uses those values to determine what action to take and then assigns values to the object's attributes to convey information back to the Windows control. The **`TvCustomDrawSimple`** object is specific to the **`TreeView`** class. Other ooDialog controls that support custom draw have their own class that performs a similar function, but whose attributes are specific to that control. For instance, the *ListView* class uses the *LvCustomDrawSimple* for its CUSTOMDRAW event handler.

26.54.1. CustomDraw Event Handler

When the tree-view control is registered for *simple* custom draw, the event handler is invoked when the draw stage is item prepaint. For simple custom draw, the ooDialog framework handles the other draw stages internally. It makes the appropriate response to the dialog control so that the dialog control will send the item prepaint notifications.

The single argument passed to the event handler is a **`TvCustomDrawSimple`** object. The event handler examines the values of the attributes of the object to determine the information sent by the underlying tree-view control. The event handler then assigns values to the object that specify how the tree-view control should draw the item. Finally, the programmer returns true to have the updated information passed on to the tree-view, or false to have the tree-view draw the item as it normally would.

```
::method onCustomDraw unguarded
  use arg tvcdSimple

  return boolean
```

Arguments:

The event handling method receives 1 argument:

tvcdSimple

A **`TvCustomDrawSimple`** object whose attributes are used to convey information back and forth between the underlying tree-view control and the event handler.

Return:

Return true to indicate that the attributes in the **`TvCustomDrawSimple`** object are to be used by the tree-view when drawing the item. Return false to indicate the tree-view should draw the item itself. Returning false is the equivalent of using the *CDRF_DODEFAULT* value for the response to the tree-view.

Remarks:

The key to using custom draw is to examine the attributes of the **`TvCustomDrawSimple`** object sent as the argument to the event handler to determine which item is about to be drawn, and then set attributes in the object to customize the drawing of that item.

As an example, say the goal was to draw every other level in the tree-view with a different color. The event handler will be invoked before each item is to be painted. The programmer would examine the *level* attrbute to determine if it was an odd or even level. If the level was even, the color attributes, *clrTextBk* and *clrText*, would be set to the desired colors to color the item. If the level was odd, the event handler would set the color attributes to the colors for a item at an odd level. Then the *reply* attribute would be set to a response value that indicates to the tree-view is to use the information in the object to draw the item, and the event handler would return true to indicate that custom drawing is to be done. The example below demonstrates this:

Example

The following example shows the code for the discussion in the remarks section

```
::method onCustomDraw unguarded
  use arg tvcds

  if tvcds~level // 2 == 1 then do
    tvcds~clrText   = self~RGB(82, 61, 0)
    tvcds~clrTextBk = self~RGB(250, 250, 250)
  end
  else do
    tvcds~clrText   = self~RGB(0, 20, 82)
    tvcds~clrTextBk = self~RGB(250, 250, 250)
  end

  tvcds~reply = self~CDRF_NEWFONT
  return .true
```

26.54.2. Method Table

The following table lists the class and instance methods of the **`TvCustomDrawSimple`** class:

Table 26.2. TvCustomDrawSimple Class Method Reference

Method	Description
Class	**Methods**
new	The Rexx programmer can not instantiate a **`TvCustomDrawSimple`**object, the ooDialog framework instantiates these objects
Attribute	**Methods**
clrText	Reflects a custom color that the text of the tree-view item should be drawn with.
clrTextBk	Reflects a custom color that the text background of the tree-view item should be drawn with.
drawStage	Reflects the draw *stage* of the current paint cycle.
font	Reflects a custom font that the tree-view should use for the text.
id	Reflects the resource ID of the tree-view control sending the custom draw event notification message.
item	Reflects the *handle* of the tree-view item that needs to be drawn.
itemData	Reflects the item *item* for the tree-view item that needs to be drawn.
level	Reflects the level in the tree of the item that needs to be drawn.
reply	Reflects the *CDRF_** response value that is used to reply to the event notification.

26.54.3. new (Class Method)

```
>>--new-----------------------------------------><
```

The **`TvCustomDrawSimple`** class can not be instantiated by the Rexx program. When needed, the ooDialog framework instantiates a **`TvCustomDrawSimple`** object and passes it to the dialog's custom draw event handler TODO need link.

26.54.4. clrText (Attribute)

```
>>--clrText-----------------------------------------------------><

>>--clrText-=-varName-------------------------------------------><
```

Reflects a custom color that the text of the tree-view item should be drawn with. This attribute is information sent back to the tree-view control.

clrText get:
This is a set-only attribute, it is information to be sent back to the tree-view control.

clrText set:
Set this attribute to the color value for the text of the item.

Remarks:
To construct the proper color value use the *rgb* method of the **`CustomDraw`** class. Note that the *CLR_DEFAULT* value does not seem to work with tree-views. Experimentation seems to show that the text color must be explicitly set to the proper value.

Details
Raises syntax errors when incorrect usage is detected.

Example:
In this example snippet of code the *clrText* attribute is set to a reddish color and the background color is set to a yellowish color:

```
    cdInfo~clrText   = self~RGB(247,   7,  59)
    cdInfo~clrTextBk = self~RGB(245, 245, 127)
```

26.54.5. clrTextBk (Attribute)

```
>>--clrTextBk-----------------------------------------------------><

>>--clrTextBk-=-varName-------------------------------------------><
```

Reflects a custom color that the text background of the tree-view item should be drawn with. This attribute is information sent back to the tree-view control.

clrTextBk get:
This is a set-only attribute, it is information to be sent back to the tree-view control.

clrTextBk set:

Set this attribute to a custom color that should be used for the text background when the item is drawn.

Remarks:

To construct the proper color value use the *rgb* method of the **`CustomDraw`** class. Note that the *CLR_DEFAULT* value does not seem to work with tree-views. Experimentation seems to show that the text color must be explicitly set to the proper value.

Details

Raises syntax errors when incorrect usage is detected.

Example:

In this example snippet of code the *clrText* attribute is set to a reddish color and the background color is set to a yellowish color:

```
cdInfo~clrText   = self~RGB(247,   7,  59)
cdInfo~clrTextBk = self~RGB(245, 245, 127)
```

26.54.6. drawStage (Attribute)

```
>>--drawStage-----------------------------------------------------><

>>--drawStage-=-varName-------------------------------------------><
```

Reflects the draw *stage* of the current paint cycle. This attribute is information sent from the tree-view control.

drawStage get:

Returns the draw stage constant value as sent by the tree-view control.

drawStage set:

This is a get-only attribute. It is information sent from the tree-view control and can not be changed.

Remarks:

In *simple* custom draw, the ooDialog framework handles many of draw stage notifications internally and the event handler only gets invoked for the *CDDS_ITEMPREPAINT* draw stage.

Details

Raises syntax errors when incorrect usage is detected.

26.54.7. font (Attribute)

```
>>--font-----------------------------------------------------><

>>--font-=-varName-------------------------------------------><
```

Reflects a custom font that the tree-view should use for the text. This attribute is information sent back to the tree-view control.

font get:

This is a set-only attribute, it is information to be sent back to the tree-view control.

font set:

Set this attribute to the *handle* of a font to used for the text of the item.

Remarks:

To have the font drawn in the default font of the tree-view simply do not set the *font* attribute to anything. For example, if the color for the font should be changed, but not the font itself, then the programmer should not set this attribute.

Use the *createFontEx* method to get the handle of a font. Microsoft recommends that the response code when a new font is to be used include *CDRF_NEWFONT* value.

Details

Raises syntax errors when incorrect usage is detected.

26.54.8. id (Attribute)

```
>>--id-----------------------------------------------------><

>>--id-=-varName-------------------------------------------><
```

Reflects the resource ID of the tree-view control sending the custom draw event notification message. This attribute is information sent from the tree-view control.

id get:

Returns the numeric *resource ID* of the tree-view control that sent the custom draw event notification.

id set:

This is a get-only attribute. It is information sent from the tree-view control and can not be changed.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example uses custom draw with a list-view and a tree-view. It uses the same event handler, *onCustomDraw* for both controls and then use the *id* attribute to determine which control has sent the event notification:

```
  self~customDraw
  self~customDrawControl(IDC_LV_STATS, 'ListView', onCustomDraw)
  self~customDrawControl(IDC_TV_PLAYERS, 'TreeView', onCustomDraw)

  ...

::method onCustomDraw unguarded
    use arg customDrawInfo

    if customDrawInfo~ID == .constDir[IDC_LV_STATS] then
        return self~customDrawStats(customDrawInfo)
    else
        return self~customDrawPlayers(customDrawInfo)
```

26.54.9. item (Attribute)

```
>>--item-----------------------------------------------------><

>>--item-=-varName-------------------------------------------><
```

Reflects the *handle* of the tree-view item that needs to be drawn. This attribute is information sent from the tree-view control.

item get:

Returns the item handle that the event notifications is for.

item set:

This is a get-only attribute. It is information sent from the tree-view control and can not be changed.

Remarks:

Use the item index to determine exactly which item or subitem is going to be painted.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example changes the font and colors of the selected item in a tree-view It uses the *item* attribute to determine if item to be drawn is the selected item or not. Notice that if the item is not the selected item, .false is returned which tells the tree-view to draw the item without any customization:

```
    bkClr        = self~RGB(250, 250, 250)
    selectedClr  = self~RGB(82, 0, 20)
    selectedFont = self~createFontEx('Courier New', 10)

    self~customDraw
    self~customDrawControl(IDC_TREE, 'TreeView')
    ...

::method onCustomDraw unguarded
    expose tv bkClr selectedClr selectedFont
    use arg tvcds

    if tv~selected == tvcds~item then do
      tvcds~clrText   = selectedClr
      tvcds~clrTextBk = bkClr
      tvcds~font      = selectedFont
      tvcds~reply     = self~CDRF_NEWFONT
      return .true
    end

    return .false
```

26.54.10. itemData (Attribute)

```
>>--itemData-------------------------------------------------><

>>--itemData-=-varName---------------------------------------><
```

Reflects the item *data* for the tree-view item that needs to be drawn. This attribute is information sent from the tree-view control.

itemData get:

Returns the item data object for the item about to be drawn, or `.nil` if no item data was set for the item.

itemData set:

This is a get-only attribute. It is information sent from the tree-view control and can not be changed.

Remarks:

The tree-view allows the programmer to associate a value with any tree-view item. This item data value is sent by the tree-view in the event notification information. The programmer can set the item data for a tree-view item in several ways, using the *setItemData* of the *TreeView* class, for example.

The *itemData* attribute could be used to help decide how to custom draw the tree-view item or subitem, see the example below.

Details

Raises syntax errors when incorrect usage is detected.

Example:

In this example the color to draw the text for each item is stored in the item data for the item. This makes for a very simple event handler, the text color is just set to whatever value the item data is:

```
::method onCustomDraw unguarded
    use arg tvcds

    itemData = tvcds~itemData

    tvcds~clrText   = itemData~textColor
    tvcds~clrTextBk = itemData~backgroundColor
    tvcds~reply     = self~CDRF_NEWFONT

    return .true
```

26.54.11. level (Attribute)

```
>>--level-------------------------------------------------------><

>>--level= varName---------------------------------------------><
```

Reflects the level in the tree of the item that needs to be drawn. This attribute is information sent from the tree-view control.

level get:

Returns the level in the tree of the item to be drawn. The root of the tree is level 1. All direct children of the root are at level 2. A direct child of any node at level 2 is at level 3, and so on.

level set:

This is a get-only attribute. It is information sent from the tree-view control and can not be changed.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This snippet of code colors each level in a tree in alternating colors:

```
...

if tvcds~level // 2 == 1 then do
  tvcds~clrText   = oddLevelClr
  tvcds~clrTextBk = bkClr
end
else do
  tvcds~clrText   = evenLevelClr
  tvcds~clrTextBk = bkClr
end

tvcds~reply = self~CDRF_NEWFONT
return .true
```

26.54.12. reply (Attribute)

```
>>--reply-----------------------------------------------------><

>>--reply-=-varName-------------------------------------------><
```

Reflects the *CDRF_** response value that is used to reply to the event notification. This attribute is information sent back to the tree-view control.

reply get:

This is a set-only attribute, it is information to be sent back to the tree-view control.

reply set:

The *reply* attribute should be set to one of the CDRF_* constant values.

Remarks:

The *reply* attribute can be set to any valid combination of CDRF_* values. However, in *simple* custom draw only a two values make any sense. These are CDRF_NOTIFYITEMDRAW, and CDRF_NEWFONT.

Rather than set *reply* to CDRF_DODEFAULT and returning true from the event handler, the programmer should just return false from the event handler. Using any of the other CDRF_* values, in simple custom draw, will have no effect in the Rexx program and will cause the tree-view control to send unnecessary event notifications.

The only combination of CDRF_* values that make sense is combining CDRF_NEWFONT and CDRF_NOTIFYITEMDRAW. And, this is never really needed. Using CDRF_NEWFONT is sufficient if the font is changed and CDRF_CDRF_NOTIFYITEMDRAW is sufficient if any of the colors have been changed.

Details

Raises syntax errors when incorrect usage is detected.

Up Down Controls

Up-down controls consist of a pair of arrow buttons that the user can click to increment or decrement a value, such as a scroll position or a number displayed in a companion control. The value associated with an up-down control is called its current position. An up-down control is most often used with a companion control, which is called a buddy window.

To the user, the up-down control with its buddy window often look like a single control. The programmer can specify that an up-down control automatically position itself next to its buddy window and that it automatically set the text of the buddy window to its current position. For example, an up-down control with an edit control can be used to prompt the user for numeric input. Sometimes an up-down control with an edit control for its buddy window is referred to as a spinner control.

Up-down controls can be used without a buddy window to provide a simplified type of scroll bar. Up-down controls allow the programmer to change the radix base used to display the current position of the up-down control from 10 to 16. The controls also support acceleration. The programmer can control the rate at which the position changes when the user holds down an arrow button. The longer the user holds down the button, the faster the up-down control changes postion.

For an up-down control with a buddy window, the current position is the number in the buddy window's text. When the programmer queries the up-down control for its position, this text may have changed. For example, if the buddy window is an edit control, the user may have manually typed in a number in the edit control. Therefore, when the up-down control is queried for its position, it first retrieves the current text from the buddy window and updates its position using that text.

The possible range of positions for an up-down control is -2,147,483,648 to 2,147,483,647. However, the range for any specific up-down control can be set by the programmer. The maximum value of the range can be less than the minimum range. In this case the up arrow will reduce the current position, the down arrow increases the current position. In other words up means moving towards the maximum and down means moving towards the minimum.

The **UpDown** class provides methods to work with and manipulate the underlying Windows up-down dialog control which it represents. It is a concrete subclass of the dialog *control* object and therefore has all methods of the of the dialog control object.

In addition to the methods of the class itself, the following methods from other classes in the ooDialog framework are needed, or are useful, when working with up-down controls:

Instantiation:

Use the *newUpDown* method of the *dialog* object to retrieve a new UpDown object.

Dynamic Definition:

To dynamically define an up-down control in a *UserDialog* class, use the *createUpDown* method.

Event Notification

To connect the *event* notifications sent by the underlying up-down control to a method in the Rexx dialog object use the *connectUpDownEvent*() method.

27.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with UpDown objects, including the pertinent methods from other classes:

Table 27.1. Important UpDown Methods

Method	Description
Useful	**External Methods**

Method	Description
newUpDown	Returns an **UpDown** object for the control with the specified ID.
createUpDown	Creates an up-down control in the dialog template of a *UserDialog*
connectUpDownEvent	Connects up-down event notifications to a method in the Rexx dialog object
Class Methods	**Class Methods**
deltaPosReply	Constructs the proper return object for the event handler of the up-down DELTAPOS event.
Instance Methods	**Instance Methods**
getAcceleration	Retrieves the acceleration information for the up-down control in an array of **Directory** objects.
getBase	Retrieves the current base, either base 10 or 16, for the up-down control.
getBuddy	Retrieves the up-down's buddy window as a dialog control object.
getPosition	Returns the position of the up-down control as a signed whole number.
getRange	Returns the current range of the up-down control in a **Directory** object.
setAcceleration	Sets the acceleration for the up-down control.
setBase	Sets the radix base for the up-down control.
setBuddy	Sets the buddy window for the up-down control.
setPosition	Sets the current position of the up-down control.
setRange	Sets the range of positions for the up-down control to that specified.

27.2. newUpDown (dialog object method)

UpDown objects can not be instantiated by the programmer from Rexx code. Rather an UpDown object is obtained by using the *newUpDown* method of the *control* object. The syntax is:

```
>>-newUpDown(--id--)-----------------------------><
```

27.3. createUpDown (UserDialog method)

An UpDown object can be added to the dialog template for a *UserDialog* dialog through the *createUpDown* method. The basic syntax is:

```
>>--createUpDown(-id-,--x-,--y-,--cx-,--cy-+---------+--+-----------------+--)---><
                                           +-,-style-+  +-,-attributeName-+
```

27.4. connectUpDownEvent (dialog object method)

To connect event notifications from an up-down object use the*connectUpDownEvent* method of the *dialog* object. The basic syntax is:

```
>>-connectUpDownEvent(--id--,--event--+---------------+--)------------><
                                      +--,-methodName--+
```

27.5. deltaPosReply (Class method)

```
>>--deltaPosReply(--+----------+--+----------+--+-------------+--)-------------><
                    +--change--+  +-,-cancel-+  +-,-newDelta--+
```

Constructs the (proper) reply object for a *DELTAPOS* up-down *event* notification.

Arguments:

The arguments are:

change [optional]

If true, the change in the position of the up-down control is to be canceled or modified. If false, then the proposed change in position is allowed. When this argument is false, the other arguments are always ignored. The default is false.

cancel [optional]

If true, the delta position message is canceled completely. If false the message is not canceled, and the delta position is set to *newDelta*. The default is false.

newDelta [optional]

The amount by which the current position of the up-down control is changed. The default is 1. This argument is only used if *change* is true and *cancel* is false.

Return value:

The return is a position change reply buffer that is suitable to use as the return from the event handler for the DELTAPOS event notification from an up-down control.

Remarks:

A position reply buffer can not be constructed by the Rexx programmer, so the return from this method must be used as the return from the event handler for the DELTAPOS event.

Details

Raises syntax errors when incorrect arguments are detected.

27.6. getAcceleration

```
>>--getAcceleration------------------------------><
```

Retrieves the acceleration information for the up-down control in an array of **`Directory`** objects.

Arguments:

This method takes no arguments.

Return value:

An array of **`Directory`** objects. Each directory contains the following indexes:

SECONDS

The amount of elapsed time, in seconds, before the position change increment specified by the *increment* index is used.

INCREMENT

The increment used to change the position after the time specified by the *seconds* index elapses.

Remarks:

Acceleration information allows the up-down control change its position faster the longer the user holds down the control's up or down arrow, or the up or down keyboard arrow. A single acceleration specifies that after so many seconds increment the position by this amount. For example, after 1 second increment the postion by 4. After 2 seconds increment the postion by 8.

The accelerations are always sorted by seconds in ascending order. Typically the first acceleration would be 0 seconds with an increment of 1. The programmer can change the acceleration information for the up-down control through the *setAcceleration* method.

27.7. getBase

```
>>--getBase--------------------------------------><
```

Return the current base (the radix base, either base 10 or 16) for the up-down control.

Arguments:

There are no arguments for this method.

Return value:

The current base, either 10 or 16, for the up-down control.

27.8. getBuddy

```
>>--getBuddy-------------------------------------><
```

Retrieves the current buddy window, as a dialog control object, of the up-down, or `.nil` if the up-down does not have a buddy window.

Arguments:

This method does not take any arguments.

Return value:

The return is the dialog *control* object that is the buddy window of the up-down control. Normally, or at least most often, this will be an *Edit* object. However, it could be some other type of dialog control object. If the up-down control does not have a buddy window, then `.nil` is returned.

Example:

It is difficult to prevent the user from typing in numbers, that are outside of the range of the up-down, in an edit control that is the buddy window. This example shows a generic method that could be used to check that what the user typed lies within the proper range of the up-down.

```
::method checkUpDownIntegrity private
  use strict arg upd

  buddy = upd~getBuddy
  if buddy \== .nil then do
    pos = upd~getPosition
    if buddy~getText \== pos then buddy~setText(pos)
  end
```

27.9. getPosition

```
>>--getPosition----------------------------------><
```

Returns the current position of the up-down control.

Arguments:

There are no arguments to this method.

Return value:

The return value is the current position of the up-down control.

Remarks:

When the *getPosition* method is invoked, the up-down control updates its position based on the text of its buddy window. If there is no buddy window, or if the text is outside of the range of the up-down control, then the position is not changed and **`.systemErrorCode`** is set to 13, the data is invalid.

Details

Sets the *.systemErrorCode*. See the remarks section.

Example:

This example retrieves the up-down's position value during validation when the dialog is closing. If the user has manually typed in a value in the buddy edit control, then that is the value that will be used. However, if that value is out of range, the validation is failed and the user is warned:

```
::method validate unguarded

  ...
  upd = self~newUpDown(IDC_UPD_YEARS_EXPERIENCE)
  years = upd~getPostion
  if .systemErrorCode <> 0 then do
    eol = .endOfLine
    edit = upd~getBuddy

    msg = 'You've typed in' edit~getText 'years of' || eol || -
          'experience.  That is outside of the'     || eol || -
          'allowable range.'               || eol~copies(2) || -
          'Please use a reasonable number.'

    title = "Job Application: Software Engineer"
    button = "OK"
    icon = "INFORMATION"
    miscStyles = "APPLMODAL TOPMOST"
    j = messageDialog(msg, self~hwnd, title, button, icon, miscStyles)

    edit~setText(years)
    return .false
  end

  return .true
```

27.10. getRange

```
>>--getRange-------------------------------------><
```

Returns the current range of the up-down control in a **`Directory`** object.

Arguments:

There are no arguments to this method.

Return value:

A **`Directory`** object containing the minimum and maximum position for the up-down control. The directory contains the following indexes:

MIN

The minimum value of the current range for the up-down control.

MAX

The maximum value of the current range for the up-down control.

Remarks:

The maximum range value can be less than the minimum value. The up arrow moves towards the maximum and the down arrow moves towards the minimum. If the maximum value is less than the minimum, then using the up arrow will decrease the current position and the down arrow will increase the positions.

27.11. setAcceleration

```
>>--setAcceleration(--accelerations--)-----------><
```

Sets the acceleration for the up-down control.

Arguments:

The single argument is:

accelerations

An array of **`Directory`** objects. Each directory in the array defines a single acceleration and must contain the following indexes:

SECONDS

The amount of elapsed time, in seconds, before the position change increment specified by the *increment* index is used.

INCREMENT

The increment used to change the position after the time specified by the *seconds* index elapses.

The array can not be sparse, that is each index must contain a value. The value at each index must be a **`Directory`** object and must contain the proper indexes.

Return value:

On succes, true. On failure false.

Remarks:

Acceleration information allows the up-down control change its position faster the longer the user holds down the control's up or down arrow, or the up or down keyboard arrow. A single acceleration specifies that after so many seconds increment the position by this amount. For example, after 1 second increment the postion by 4. After 2 seconds increment the postion by 8, etc..

The accelerations must be sorted in the array, by seconds, in ascending order. Typically the first acceleration would be 0 seconds with an increment of 1.

Details

Raises syntax errors when incorrect usage is detected.

Example:

The following example sets the acceleration to be very fast, much faster than normal:

```
::method initDialog

  upd = self~newUpDown(IDC_UPD_HUGH_RANGE)

  accels = .array~new(5)
  do i = 0 to 4
    d = .directory~new
    d~seconds = i
    d~increment = 8 ** i
    accels[i + 1] = d
  end

  upd~setAcceleration(accels)
```

27.12. setBase

```
>>--setBase(--radixBase--)-----------------------><
```

Sets the radix base for an up-down control.

Arguments:

The single argument is:

radixBase

The base for the up-down. Can be 10 or 16.

Return value:

Returns the previous base of the up-down control, or 0 if the new base specified is not valid.

Remarks:

The base value determines how the numbers are displayed in the buddy window. With base 10, the numbers displayed are decimal digits, and with base 16, hexadecimal digits. Hexadecimal numbers are always non-negative, (not signed,) and decimal numbers are whole numbers, (they can be signed.)

Example:

This example changes the base of the up-down control to 16:

```
  upDown = self~newUpDown(IDC_UPD)
  ret = upDown~setBase(16)
  if ret == 16 then say "The up-down base was already set to 16."
  else if ret == 10 say "The up-down base was changed from 10 to 16."
  else say "This else will never execute."
```

27.13. setBuddy

```
>>--setBuddy(--dlgControlObj--)------------------><
```

Sets the buddy window for the up-down control.

Arguments:

The single argument is:

dlgControlObj [required]

The dialog control object that is to be the new buddy window for the up-down control.

Return value:

Returns the previous dialog control object of the buddy window, if there was a previous buddy. Returns **.nil** if there was no previous buddy window.

Remarks:

When an up-down control has the AUTOBUDDY style, the up-down control selects the previous window in the dialog template as its buddy window. If the up-down control does not have that style, the *setBuddy* method can be used to assign a specific dialog control as the buddy window. Of course, the *setBuddy* method can be used to assign a specific dialog control, when desired, for any other reason the programmer may have.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example sets the buddy window during the *initDialog* method for an up-down control that does not have the AUTOBUDDY style.

```
::method initDialog

  edit = self~newEdit(IDC_EDIT_EXPIRATION_YEAR)
  upd = self~newUpDown(IDC_UPD_EXPIRATION)
  upd~setBuddy(edit)
```

27.14. setPosition

```
>>--setPosition(--newPosition--)-----------------><
```

Sets the position of the up-down control.

Arguments:

The single argument is:

newPosition [required]

A whole number in the range of -2,147,483,648 to 2,147,483,647 that specifies the new position for the up-down control.

Return value:

The previous position of the up-down control.

Remarks:

If *newPosition* is outside the control's current *setRange*, the position is set to the nearest valid position within the range.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example sets the range for an up-down control to be the years 1900 through 2100 and then sets the current position of the up-down control to the current year:

```
::method initDialog

  upDown = self~newUpDown(IDC_UPD_YEARS)
  upDown~setRange(1900, 2100)
  upDown~setPosition(.DateTime~today~year)
```

27.15. setRange

```
Form 1:

>>--setRange(--+----------------+--)-------------><
               +--directoryObj--+

Form 2:

>>--setRange(--+-------+-+--------+--)-----------><
               +--min--+ +-,-max--+

Generic form:

>>--setRange(--+---------+--)--------------------><
               +--range--+
```

Sets the range of positions for the up-down control to that specified.

Arguments:

The *range* argument(s) are optional. If omitted, the range is set to a minimum of 0 and a maximum of 100.

Form 1:

The single argument for form 1 is:

directoryObj [optional]

A **`Directory`** object whose indexes specify the minimum and maximum range. The directory can contain the following indexes:

MIN [optiona]

The minimum value for the range. If this index is omitted, the minimum is set to 0.

MAX [optional]

The maximum value for the range. If this index is omitted, the maximum is set to 100.

Form 2:

The arguments for form 2 are:

min [optiona]

The minimum value for the range. If this argument is omitted, the minimum is set to 0.

max [optional]

The maximum value for the range. If this argument is omitted, the maximum is set to 100.

Return value:

This method always returns 0.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example sets the range of an up-down control when the dialog first starts. Later in the life cycle of the dialog the range is changed numerous times, so a generic functions is used to change the range:

```
::method initDialog
  expose upDown staticPresident

  upDown = self~newUpDown(IDC_UPD_LIFE_SPAN)
  upDown~setRange(1732, 1799)

  staticPresident = self~newStatic(IDC_ST_PRESIDENT)
  static~President~setText("George Washington")

  ...

::method onComboBoxSelect
  expose staticPresident

  president = self~newComboBox(IDC_CB_PRESIDENT)~selected
  staticPresident~setText(president)

  select
    when president == 'Abraham Lincoln' then do
      r = .directory~new
      r~min = 1809
      r~max = 1865
      self~changeRange(r)
    end
    when president == 'Andrew Jackson' then do
      r = .directory~new
      r~min = 1767
      r~max = 1845
      self~changeRange(r)
    end

    ...

    when president == 'Zachary Taylor' then do
      r = .directory~new
      r~min = 1784
      r~max = 1850
      self~changeRange(r)
    end
  end

::method changeRange private
  expose upDown
  use strict arg range

  upDown~setRange(range)
```

Menus

Menus are lists of items that specify options for an application. Groups of related options are usually placed in submenus. Clicking a menu item either opens a submenu or causes the application to carry out a command.

Menus are arranged as a hierarchy. A *menu bar* is the top of the hiearchy for menus attached to an application window. The menu bar will contain a list of *menus*, and menus can contain *submenus*. Menu bars are often called *top-level* menus, while the menus and submenus are also known as *pop-up* menus.

A menu *item* either carries out a command, opens a submenu, or is special type of menu item called a *separator*. Menu items that do not open a submenu are called *command* items or simply commands. Menu items that open a submenu are sometimes called *menu names*. Separators appear as a horizontal line. They are used to divide a menu into groups of related items. A separator cannot be used in a menu bar, and the user cannot select a separator.

Only top-level windows can have menu bars. Child windows can not have a menu bar. If the window has a title bar the system positions a menu bar underneath it. All menus must have an owner window.

The system also provides *shortcut menus* and the *window* menu. A shortcut menu is not attached to a menu bar and can appear anywhere on the screen. Shortcut menus are typically activated by the user right-clicking on a specific portion of the screen and the menu items are usually only relevant to what the user clicked on. For this reason, shortcut menus are often called *context* menus. The window menu is also knows as the *System* menu or the *Control* menu. The System menu is usually managed exclusively by the system. The user activates the System menu by clicking on the icon in the upper left corner of a window, or by right-clicking anywhere on the title bar. Unlike menu bars, child windows can have a System menu. It is unusual for a top-level window to not have a System menu.

The menu related classes in the following table are described in this chapter. The primary classes that the ooDialog programmer will use are the **`BinaryMenuBar, UserMenuBar, ScriptMenuBar`**, and **`PopupMenu`**. The other classes are mixin classes that implement the functionality common to all types of menus. The binary, user, and script menu bar classes all have the same essential functionality. They just differ in how they are initially created. The popup menu class is similar to the menu bar classes, it just does not have any methods that are only relevant to a menu bar.

Table 28.1. ooDialog Menu Classes

Class	Description
BinaryMenuBar	A menu bar created from a menu resource, or created as an empty menu bar.
Menu (Mixin Class) The Menu *Menu*	The basis for the menu object.
MenuBar (Mixin Class) The Menu Bar *MenuBar*	The basis for the menu bar object.
PopupMenu	A menu or a shortcut menu, also known as a submenu or a context menu.
ScriptMenuBar	A menu bar created from a resource script file.
SystemMenu	A system menu, also known as the window menu or the control menu.
UserMenuBar	A menu bar created dynamically by the programmer in a fashion similar to a *UserDialog*.

28.1. Menu Item IDs

Each menu item in a menu can be uniquely identified. This documentation generically refers to a menu item *ID*, but there are really two ways of identifying a menu item.

Menu Item Identifier

Each menu item has a unique, positive, integer assigned to it by the programmer, called a menu item identifier. These numbers are essentially the same as any other *resource ID* used in ooDialog. The Rexx programmer can use numeric or *symbolic* IDs for menu item identifier's, in the same way as for other resource IDs.

All menus assign menu item identifiers to command items and submenus. ooDialog uses what the Windows operating system calls *extended* menus. Extended menus also assign identifiers to separators. Identifiers are used for *menu command event*s. In addition, a menu item can be specified using its identifier in method invocations. For example, to *enable* or *disable* a menu item.

Identifiers allow the programmer to specify any menu item anywhere in the menu hiearchy. The *getID*() method can be used to retrieve the identifier of the menu item at a specified position

Menu Item Position

In addition to having a unique identifier, each menu item in a menu bar or menu has a unique position value. Although the operating system uses zero-based position numbering, ooDialog uses one-based position numbering to conform to general Rexx practice. The leftmost item in a menu bar, or the top item in a menu, has position one. The position value is incremented for subsequent menu items. A position value is assigned to all items in a menu, including separators.

Position values can be used in method invocations that require specifying a menu item. Note that since the top item in a menu, or the leftmost menu item in a menu bar, always has a position of one, it is not possible to uniquely specify any menu item in a menu hiearchy. Rather, you can only use a position to specify an item in a specific submenu.

Symbolic Menu Item IDs

Menu *item identifiers* are essentially resource *IDs*. The programmer can use *symbolic* IDs for menu item identifiers rather than numeric IDs for any argument in a menu object method that uses a menu item identifier. However, to do so, the programmer *must* use the global *.constdir*. If the global `.constDir` is not used, then all menu item identifier arguments **must** be numeric.

Except in a few methods where it simply does not make any sense, all menu methods that require specifying a menu item, accept either a menu item identifier or a menu item position. The menu item specifier is referred to generically as the *menu item ID*. In ooDialog, all *menu item IDs* must be greater than 0. Since both identifiers and positions are numbers, the programmer needs to specify which is being used. However, by default, the ID is assumed to be an identifier. Therefore, the programmer only need indicate when the ID is a position ID.

28.2. Menu Command Items

Menu items that are not separators and do not open a submenu are called menu command items.

28.2.1. Menu Command Events

When a user selects a menu command item, an *event* is generated and the operating system sends a notification to the owner window of the menu. The ooDialog framework provides methods to connect this menu command notification to a method in the Rexx dialog object in a similar manner to connecting dialog control *notifications*. The event connection methods are contained within the various menu objects and are similar to the event connection methods of the dialog object.

However, there are some specific details when dealing with menus that should be understood. The menu command event is connected using the menu item *ID* of the command item, just as dialog control events are connected using the resource ID of the dialog control. Event notifications are always connected to a specific Rexx dialog object, the dialog object that represents the *underlying* dialog that the notification is expected to be sent to.

A Rexx dialog control object has one, and only one, underlying dialog. But, a Rexx menu *MenuBar* object can be attached to one dialog, then detached and attached to a different dialog. Likewise, the same *PopupMenu* menu could be used by any number of different dialogs. Because of this, it may be necessary to connect a menu command item notification to more than one dialog.

For example, the same context menu object could be used by the programmer in several different dialogs. In this case, it is not sufficient to connect each of the menu items once, to one dialog. Rather, the menu items need to be connected to each dialog where the context menu is used. When the programmer *show* the context menu, it will have one and only one owner dialog. When the user selects a menu item, the notification is sent to that owner dialog.

Another example would be a program that shows different dialogs at various times. The dialogs are different, but all have a menu bar that contains the same items. One approach would be to create a single menu bar, that is attached to each dialog when it is shown. The menu items of the menu bar would then need to be connected to each dialog that the menu bar is attached to during the program.

One last point the programmer should be aware of. Both a regular menu item selection and a *Button* click generate the same event notification. **Note** that a *system* menu command event uses a different type of notification than buttons or regular menus. A system menu command event notification will never be the same as a button click notification.

Sometimes the action to take when a button is clicked and the action to take when a menu item is selected are the same. In these cases, it is a common practice in Windows programming to use the same resource ID for both the button and the menu item and then use the same event handler for both the button push and the menu item selection. To use this method in ooDialog, the programmer would need to only connect the event once.

E.g., take the case of an OK push button and an Exit menu item. Many times the action to take for both is exactly the same. Here the ooDialog programmer would use the same resource ID for both the button and the menu item. OK buttons normally use the resource ID of **`IDOK`**, which the ooDialog framework *automatically* connects to the *ok* method. If the programmer also uses the IDOK id for the Exit menu item, then nothing else need be done. Whether the user selects the Exit menu item, or clicks the OK button, the same method in the Rexx dialog object will be invoked.

Notice that, since a button click and a menu item selection generate the same notification, it is not possible to connect a menu item and a button click to separate methods if they both have the same resource ID.

28.2.2. Menu Command Event Connections

Menus are separate objects from dialog objects and can be instantiated with no reference to any dialog. In addition, menus can be assigned different owner dialogs during an application's life time. Because of this, unlike connecting dialog control envents, there are several different strategies that can be used when connecting menu command events.

The menu classes provide a number of ways, and a number of methods, to connect menu command events to methods in the Rexx dialog. There are three basic types of ways to make command event connections, *auto connection*, *connect request*, and *direct connection*.

Auto Connection

When auto connection is on, *each* time a menu bar is attached to a dialog, or a context menu is assigned to an owner dialog, every menu command item is automatically connected to a method in that dialog. The programmer can specify if every command item is connected to the same method, or if every command item is connected to a unique method. When every command item is connected to an unique method, ooDialog automatically generates the method name based on the text of the command item.

Note that if auto connection is on, the automatic connection of menu items is *always* done when a menu bar is attached to a dialog or a context menu is assigned its owner dialog. This is true even if the *connection request* way of connecting command items is also in use.

Auto connection can be explicitly turned on by using the *setAutoConnection* *before* a menu bar is attached, or a context menu is assigned, to a dialog. *setAutoConnection* can also be used to turn auto connection off at any time.

The following is some specific details regarding auto connection for the different menu classes and menu class methods:

BinaryMenuBar class

Auto connection can also be turned on explicity in the *new* method of the **`BinaryMenuBar`**.

insertItem method

If auto connection is turned on in the menu bar that the item is inserted into, and the menu bar is already attached to a dialog, then the item will be automatically connected. The method name that is used for the connected method is based on the current auto connection status. *getAutoConnectStatus* can be used to determine the current auto connection status.

This will *only* be done if the item is being inserted by item identifier, not if it is being inserted by item position. It is only done if the item is being inserted into a menu bar, not if it is being inserted into a popup menu.

Note that the arguments to the *insertItem* method, the *connect* argument, can also be used to specify a *connection request* type of connection. If a connection request is made using the *connect* argument and auto connection is turned on, both connections will be made. This should *not* be done, the result is non-deterministic.

PopupMenu class

When a popup menu is used as a submenu of a menu bar, auto connection is meaningless. A submenu can not be attached or assigned to a dialog. When a popup menu is used as a context menu, auto connection can be turned on, either explicitly by using the *setAutoConnection* method, or by using the *autoConnect* option in the *assignTo* method.

Note that *setAutoConnection* has to be invoked on the top-level menu of the context menu. Invoking it on a submenu of the context menu would have no effect. When the *track* or *show* methods are used with a popup menu, auto connection is ignored.

ScriptMenuBar class

Auto connection can be turned on explicitly using the *setAutoConnection* method *after* the **`ScriptMenuBar`** is instantiated, but *before* it is attached to a dialog.

UserMenuBar class

Auto connection can be turned on in the *new* method of the **`UserMenuBar`**.

Auto connection can be turned on explicitly using the *setAutoConnection* method *after* the **`UserMenuBar`** is instantiated, but *before* it is attached to a dialog.

Connection Request

A *connection request* is made when a menu command item is to be connected to a specific method name in a dialog, before the menu bar has been attached to a dialog. Obviously it is impossible to connect the item to a dialog method, before it is known what dialog the menu bar is going to be connected to. All connection requests are fulfilled the *first* time the menu bar is connected to a dialog. The requests are then discarded.

Connection requests are made by specifying a method to invoke when adding a command item to a **`UserMenuBar`** through the *addItem* method, or by specifying **`.true`** for the *connect* argument in the *new* method of the **`ScriptMenuBar`**. The request can also, possibly, be made by specifying a method to invoke when inserting a command item through the *insertItem* method.

The following is some specific details regarding connection requests for the different menu classes and menu class methods:

BinaryMenuBar class

Connection requests are not used in **`BinaryMenuBar`** class, except possibly though the *insertItem* method.

insertItem method

A connection request can be made through the *insertItem* method under these specific conditions. The *insertItem* method is invoked on a menu bar, not using by position, and the menu bar is not currently attached to a dialog. Then a connection request is made by using the last arguments to the method, the *connect* and possibly the *mName* methods. If the menu bar is already attached to a dialog, then those last two arguments connect the item immediately, which in this discussion is considered the *direct connect* type of making event connections.

Note that if auto connection is turned and a connection request is made using the *insertItem* method, then both connections will be made. This should *not* be done, the result is non-deterministic.

PopupMenu class

Connection requests have no meaning in a popup menu, there is no such thing.

ScriptMenuBar class

Connection requests are made by using the *connnect* argument in the *new* method of the class. If *connect* is true, then a connection request is made for each command item found in the resource script, using a method name constructed from the text of the command item.

UserMenuBar class

In a **`UserMenuBar`**, a connection request can be made in the *addItem* method by specifying a method name as the last argument, the *mName* argument.

As can be seen from the discussion above. the connection request type of making connections is primarily meant to allow the programmer to automatically connect the menu items in a resource script when the script is parsed (**`ScriptMenuBar`**), or when the *addItem* method is used (**`UserMenuBar`**.)

Direct Connection

The direct connection type of connecting command item events is an explicit, connect *this* command item to a method in *this* dialog. This is relatively simple to understand. Directly connecting menu command events is done through the following methods:

- *connectCommandEvent* (Class method)
- *connectAllCommandEvents*
- *connectCommandEvent* (Instance method)
- *connectSomeCommandEvents*

In general, only one type of making command event connections should be used in any one menu. Auto connection should never be used with the other two types of making connections. Mixing the 3 types of connecting command items to methods can result in unpredictable results.

- The same command item can end up connected to different methods, which method is invoked is non-deterministic.
- Numerous entries in the dialog's message table for the same command item connected to the same method can happen.
- Both of the above can result in the message table becoming far larger than necessary.

At first glance, making menu command event connections may seem a little complex. However, **in practice**, in ooDialog programs, it is relatively straight forward. The complexity only arises when multiple menus are used with multiple dialogs in a single application and the programmer *chooses* to mix the different types of making connections. *Most* ooDialog programs contain a single dialog with a single menu bar and / or a single context menu. If the programmer picks one type of making the menu item connection for her menu(s), things are really pretty simple:

```
::method initDialog
  expose menuBar

  menuBar = .BinaryMenuBar~new("menuResourceEx.dll", IDR_MENU1, , self, .true, onMenuSelect)
```

The above creates a menu bar from a binary resource, attaches it to the dialog, and automatically connects all the menu items in the menu to the *onMenuSelect* method, in one step. Which is pretty simple.

28.2.3. Menu Command Event Handlers

There are only two types of event handlers that the ooDialog programmer will need to code. Although similar, there are a few differences between the two. One *handler* is exclusively for menu command event notifications generated by the system menu. The other handler, described first, is for all other menu command event notifications.

28.2.3.1. Menu Object Command Event Handler

Non-system menu command *event* notifications are actually the same notifications as the *connectButtonEvent* CLICK event. This makes the event handler for a menu command event pretty simple. The event handler is invoked when the user selects a menu command item. The programmer should return 0 from the event handler, however the interpreter does not *wait* for the return.

```
::method onMenuSelect unguarded
  use arg id, hMenu

  return 0
```

Arguments:

The menu command event handling method receives 2 arguments:

id

The numeric menu item identifier of the command item the user selected.

hMenu

The Windows *handle* to the menu.

Return:

Return 0, although, since the interpreter does not wait for the reply, the value returned is ignored. However, it is good practice to always return a value from an event handler and 0 is the proper value for this notification.

Remarks:

Since the event handler for the menu command event is sent the ID of the menu item selected, there are two equally valid ways of coding the event handler. One way would be to connect all menu command items to one method. In that method check the menu item ID to determine which item the user selected and take the appropriate action. The other way would be to connect each menu item to its own method and ignore the arguments, simply take the action for that menu item.

Example

The following example connects the *File Open* menu item with a method and then opens a file when the user selects that menu item:

```
::method initDialog
  expose menuBar

  menuBar~attachTo(self)
  ...
  menuBar~connectCommandEvent(IDM_OPEN, onFileOpen)
  ...

::method onFileOpen unguarded
  use arg id, hMenu

  self~openFile
  return 0
```

28.2.3.2. SystemMenu Command Event Handler

System menu command *event* notifications are generated when the user clicks on a menu command item in the system menu. The interpreter *waits*) for the return from the Rexx method and the method must return either **`.true`** or **`.false`**.

```
::method onMenuSelect unguarded
  use arg id, arg2

  return .true
```

Arguments:

The menu command event handling method receives 3 arguments:

id

The identifier of the system menu command item the user selected. If this is one of the standard system menu command items, id will be a keyword identifying the item. The keywords correspond to the system menu command item *constants*, with the *SC_* removed. For example, if the menu item selected was the SC_CLOSE item, the keyword will be *CLOSE*. For non-standard system menu command items inserted into the menu by the application, *id* will be the numeric value of the item.

x

If the menu command item was selected using the mouse, *x* will be the horizontal position of the mouse cursor. When the user selects the item in some other manner, *x* will be 0.

y

If the menu command item was selected using the mouse, *y* will be the vertical position of the mouse cursor. *y* will be -1 if the item was chosen using a system accelerator, or zero if chosen using a mnemonic.

Return:

Return **`.true`** to indicate the system command was handled by the ooDialog program. Return **`.false`** to pass the command on to the operating system so that the default processing of the command item is done. For the standard system menu items, most Windows programs have the system process the selection. If the menu item selected was on inserted by the ooDialog application, then the event handler *should* return **`.true`**. Note that the Microsoft documentation says *must*.

Example

The following example instantiates a *SystemMenu* object and uses it to insert an *Edit* menu into the system menu for the application. The new menu items are connected to appropriate event handlers. Note that the event handlers shown in the example are just stubs. Real code to handle the event would need to be added:

```
::method initDialog
  expose sysMenu

  -- Get the system menu.
  sysMenu = .SystemMenu~new(self)

  -- Modify the system menu by inserting an "Edit" menu into it
  popup = .PopupMenu~new(IDM_EDIT_MENU)
  popup~insertItem(IDM_PASTE, IDM_PASTE, "Paste")
  popup~insertItem(IDM_PASTE, IDM_COPY, "Copy")
  popup~insertItem(IDM_PASTE, IDM_CUT, "Cut")

  sysMenu~insertPopup(SC_MOVE, IDM_EDIT_MENU, popup, "Edit")
  sysMenu~insertSeparator(SC_MOVE, IDM_SYS_SEP2)

  itemIDs = .set~of(IDM_COPY, IDM_CUT, IDM_PASTE)
  sysMenu~connectSomeCommandEvents(itemIDs)

  ...

::method copy
  use arg id, x, y

  say 'In SystemMenu->Edit->Copy, do a copy action id:' id 'x:' x 'y:' y
  return .true


::method cut
  use arg id, x, y

  say 'In SystemMenu->Edit->Cut, do a cut action id:' id 'x:' x 'y:' y
  return .true
```

```
::method paste
  use arg id, x, y

  say 'In SystemMenu->Edit->Paste, do a paste action id:' id 'x:' x 'y:' y
  return .true
```

28.3. The Menu Object

All the menu classes in ooDialog are similar. Menus have a large number of methods that are common to all types of menus. These common methods are contained in the **Menu** class, a mixin class inherited by all menu classes. This document treats the menu object in the same way it does the *dialog* object and the dialog *control* object, as an abstract concept. However, in this case the abstract menu object and the **Menu** mixin class are essentially the same.

The methods listed in this chapter are all methods of the **Menu** class and all ooDialog menus have these methods.

28.3.1. Method Table

The following table lists the class methods, attributes, and instance methods that all menus have in common:

Table 28.2. Menu Object Method Reference

Menu Method	Description
Class Methods	
connectCommandEvent	Connects a menu command event notification from a menu to a method in a Rexx dialog.
Attributes	
hMenu	The Windows handle of the menu.
wID	The resource ID of the menu.
Instance Methods	
check	Checks one or more menu items.
checkRadio	Checks a specified menu item in a group of items and makes it a radio item.
connectAllCommandEvents	Connects every command item in this menu to a method, or methods, in the Rexx dialog.
connectCommandEvent	Connects a menu command event notification from a menu to a method in a Rexx dialog.
connectMenuEvent	Connects menu *event* notifications, other than menu command events, to a method in the Rexx dialog.
connectSomeCommandEvents	Connects the specified subset of command items in this menu to a method, or methods, in the Rexx dialog.
deletePopup	Deletes the specified popup menu from this menu.
destroy	Releases the system resources used by this menu.
disable	Disables one or more menu items.
enable	Enables one or more menu items.

Menu Method	Description
getAutoConnectStatus	Returns a **`Directory`** object containing the current status of *autoconnection*.
getCount	Returns the number of items in this menu.
getHandle	Returns the value of the hMenu attribute of this menu.
getHelpID	Gets the Help context identifier for this menu.
getID	Gets the resource ID of the specified menu item.
getItemState	Returns a string of 0 or more keywords that indicate the current state of the specified menu item.
getMaxHeight	Gets the current maximum height for this menu, in pixels.
getMenuHandle	Retrieves the Windows *handle* of the *underlying* menu for the specified popup menu of this menu.
getPopup	Gets the specified popup menu from this menu. This menu is unchanged.
getText	Gets the text, the label, for the specified menu item. Separators do not have any text.
getType	Returns a string of 0 or more keywords that indicate the type of the specified menu item.
gray	Grays one or more menu items.
hilite	Draws one or more menu items as high lighted. A high lighted menu item is drawn as though it were selected.
insertItem	Inserts a new menu command item into this menu.
insertPopup	Inserts a new submenu into this menu.
insertSeparator	Inserts a separator into this menu at the specified position.
isChecked	Determines if the specified menu item is checked.
isCommandItem	Determines if the specified menu item is command menu item, that is if the menu item is one that a user would expect to carry out a command.
isDisabled	Determines if the specified menu item is disabled or not.
isEnabled	Determines if the specified menu item is enabled or not.
isGrayed	Determines if the specified menu item is grayed or not.
isPopup	Determines if the specified menu item is a submenu (a pop up menu.)
isSeparator	Determines if the specified menu item is a separator.
isValidItemID	Determines if a specific menu item *ID* is valid for this menu.
isValidMenu	Checks if this menu is valid.
isValidMenuHandle	Given a *handle*, determines if the handle is a menu handle and, if it is, if the menu is still valid.
itemTextToMethodName	Converts the specified text to a method name as if the text were the text of a menu command item.
releaseMenuHandle	Converts the specified text to a method name as if it where the text of a menu command item.
removeItem	Removes, but does not delete, the specified popup menu from the menu.
removePopup	Removes, but does not delete, the specified popup menu from the menu.
removeSeparator	Removes (deletes) the specified menu separator from this menu.

Menu Method	Description
setAutoConnection	Set the current status of *autoconnection*.
setHelpID	Sets the Help context identifier for the this menu, and optionally all submenus.
setID	Sets the resource ID of this menu.
setMaxHeight	Sets the maximum height for this menu, in pixels, and optionally sets the height for all submenus of this menu.
setText	Sets the text of the specified menu item..
uncheck	Unchecks one or more menu items.
unhilite	Removes the high light from one or more menu items.

28.3.2. connectCommandEvent (Class method)

```
>>--connectCommandEvent(--id-,--methodName-,--dlg--)------------><
```

Connects a menu command item *event* notification with a method in the specified Rexx dialog. The notification is sent to the Windows dialog when a command item in a menu is selected by the user.

Arguments:

The arguments are:

id [required]

The resource ID of the menu item, may be *symbolic* or numeric.

methodName [required]

The name of the method in the Rexx dialog to connect the notification to. This can not be the empty string.

dlg [required]

The Rexx dialog to connect the notification to.

Return value:

Returns **`.true`** on success, **`.false`** on error. On error the **`.systemErrorCode`** will be set as described in the Details section.

Remarks:

When using the *connectCommandEvent* class method to connect menu command events, the programmer must use a menu item *ID*, not a menu item *position* identifier.

The *connectCommandEvent* class method is used for all types of menus. However, the programmer should be aware of the differences in coding event *handler*s for *SystemMenu* and other types of menus.

Details:

Sets the *.systemErrorCode* on error. The system error code is set this way in addition to what the OS might set:

ERROR_INVALID_PARAMETER (87)

The parameter is incorrect. **Meaning:** The *methodName* argument can not be the empty string.

ERROR_WINDOW_NOT_DIALOG (1420)
The window is not a valid dialog window. **Meaning:** The *dlg* argument is not a *dialog* object.

ERROR_NOT_ENOUGH_MEMORY (8)
Not enough storage is available to process this command. **Meaning:** The dialog message table is full.

Example:

This example disables the ability of the user to move the position of the dialog at certain times. It does this by connecting the system menu's move command event to a method in the Rexx dialog. In the event handler, if the position of the dialog is *frozen*, then the method returns **`.true`** indicating that the event has been handled by the application. The operating system then takes no further action and the dialog can not be moved by the user.

The example is complete and can be copy-pasted into a program file and executed as:

```
  dlg = .FreezePositionDlg~new
  dlg~execute('ShowTop')

::requires 'ooDialog.cls'

::class 'FreezePositionDlg' subclass Userdialog

::method init
  forward class (super) continue
  self~createCenter(180, 100, 'A Freezable Dialog')

::method defineDialog
  expose isFrozen

  self~createPushButton(100,  30, 65, 55, 15, , "Take Action",   onStartAction)
  self~createPushButton(101, 100, 65, 55, 15, , "Cancel Action", onCancelAction)

  .SystemMenu~connectCommandEvent(.SystemMenu~SC_MOVE, onSCMove, self)
  isFrozen = .false

::method onSCMove unguarded
  expose isFrozen
  use arg id, x, y

  if isFrozen then return .true
  else return .false

::method onStartAction unguarded
  expose isFrozen
  isFrozen = .true

::method onCancelAction unguarded
  expose isFrozen
  isFrozen = .false
```

28.3.3. hMenu (Attribute)

```
>>--hMenu--------------------------------------------><
```

The Windows *handle* of the menu.

hMenu get:

The returned handle, if it is not null, can be used in any ooDialog method that takes a menu handle as an argument. The *getHandle*() also returns the value of this attribute.

hMenu set:

This is a *get* only attribute. The programmer can not set the attribute through assignment. I.e.,

```
menuObj~hMenu = handle
```

is not valid.

Remarks:

If a menu is released through the *destroy*() method, the menu handle will then be null and is no longer valid. The menu handle is a Rexx **`Pointer`** object and therefore has the methods of that class. Specifically it has the *isNull*() method, which can be used to test if the handle is still valid.

28.3.4. wID (Attribute)

```
>>--wID---------------------------------------------><
```

The resource ID of this menu.

wID get:

The resource ID may be -1 if the ID was not set when the menu was created.

wID set:

This is a *get* only attribute. The programmer can not set the attribute through assignment. I.e.,

```
menuObj~wID = 110
```

is not valid.

Remarks:

Although the programmer can not set the *wID* attribute directly, it can be set when menu is instantiated, for instance see the **`BinaryMenuBar`** *new* method or the **`PopupMenu`** *new* method. In addition, the *setID* method can be used to set or change the ID of a *submenu*.

28.3.5. check

```
>>--check(--IDs--+--------------+--)-------------><
                 +-,-byPosition-+
```

Checks one or more menu items. Separators can not be checked.

Arguments:

The arguments are:

IDs [required]

The item *ID(s)* to be checked, may be *symbolic* or numeric. This can be a single ID or a collection of IDs. The ID(s) can be position IDs or resource IDs, depending on the value of *byPosition*. However, if this is a collection of IDs there must be all the same type, all resource IDs or all position IDs.

byPosition [optional]

If true, the item IDs are positional IDs, otherwise they are resource IDs. The default is false, resource IDs.

Return value:

True on success, otherwise false.

Remarks:

To check more than 1 menu item at a time, use a collection object, such as an **`array`**, where each item in the collection is the menu item ID of the item to be checked.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The menu item specified to be checked is a separator.

Example:

This example is from an editor application that has a hexadecimal view of the text. The user can switch back and forth between the hexadecimal and normal views using the menu item with the text of *hex*. When the user selects the menu item, the view is toggled. A check mark by the menu item shows it is in hex view, when there is no check mark the view is normal:

```
::method initDialog
  expose inHex

  inHex = .false
  ...

::method onHexSelect unguarded
  expose menuBar inHex

  if inHex then do
    inHex = .false
    menuBar~uncheck(IDM_HEX_VIEW)
    self~switchToAscii
  end
  else do
    inHex = .true
    menuBar~check(IDM_HEX_VIEW)
    self~switchToHex
  end

  return 0
```

28.3.6. checkRadio

```
>>--checkRadio(--start-,-end-,-check--+--------------+--)-------><
                                      +-,-byPosition-+
```

Checks a specified menu item in a group of items and makes it a radio item. At the same time, the function clears all other menu items in the associated group and clears the radio-item type flag for those items.

Arguments:

The arguments are:

start [required]

The resource ID of the first menu item in the group, may be *symbolic* or numeric.

end [required]

The resource ID of the last menu item in the group, may be *symbolic* or numeric.

check [required]

The resource ID of the menu item in the group that will be checked, may be *symbolic* or numeric.

byPosition [optional]

If true, the item IDs are positional IDs, otherwise they are resource IDs. The default is false, resource IDs.

Return value:

True on success, otherwise false.

Remarks:

Note that all item IDs must be the same type, that is they must all be resource IDs or all by position IDs.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example is from an application that has a *zoom* view. The user can select a zoom in a range through the menu. A radion button menu item shows which zoom is currently in place:

```
::method setZoom
  expose menuBar
  use strict arg zoomVal

  menuBar~checkRadio(IDM_ZOOM_MIN, IDM_ZOOM_MAX, zoomVal)
  self~zoomTo(zoomVal)
  return 0
```

28.3.7. connectAllCommandEvents

```
>>--connectAllCommandEvents(--+-------+--+-------+--)----------><
                              +--mth--+  +-,-dlg-+
```

Connects every command item in the menu to a method, or methods, in the Rexx dialog.

Arguments:

The arguments are:

mth [optional]

Connect all menu command items to the method of this name. The default is to connect all menu command items to a method name composed from the text of the command item. If not omitted, *mth* can not be the empty string.

dlg [optional]
: Connect the command items to the method(s) of this dialog object. The default is to connect the command items to the owner dialog of this menu.

Return value:

Returns true on success, false on error.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)
: Incorrect function. **Meaning:** The dialog argument was omitted and the menu does not have an assigned dialog.

ERROR_NOT_ENOUGH_MEMORY (8)
: Not enough storage is available to process this command. **Meaning:** The dialog message table is full.

ERROR_WINDOW_NOT_DIALOG (1420)
: The window is not a valid dialog window. **Meaning:** The dialog argument was not ommitted, but the *dlg* is not a dialog *dialog* object.

Example:

This example connects every menu command item in the menu to the *onMenuSelect* method in the dialog:

```
::method connectMenu unguarded private
  use strict arg menu

  if \ menu~connectAllCommandEvents(onMenuSelect) then self~error('Connect All Items
 Failed')
```

28.3.8. connectCommandEvent

```
>>--connectCommandEvent(--id--,--methodName--+--------+--)---------------------><
                                             +-,-dlg--+
```

Connects a menu command item *event* notification with a method in a Rexx dialog. The notification is sent to the Windows dialog when a command item in a menu is selected by the user.

Arguments:

The arguments are:

id [required]
: The resource ID of the menu item, may be *symbolic* or numeric.

methodName [required]
: The name of the method in the Rexx dialog to connect the notification to. This can not be the empty string.

dlg [optional]
: A Rexx dialog to connect the notification to. By default, the notification is is connected to the Rexx dialog that the menu is *attachTo* or *assignTo* to. If omitted and there is no assigned or

attached dialog, no connection is made and the *.systemErrorCode* is set as described in the Details section. (To connect a menu item command event, there must be a dialog to connect it to.)

Return value:

Returns **`.true`** on success, otherwise **`.false`**. If there is an error, the **`.systemErrorCode`** is set as described in the Details section.

Remarks:

When using the *connectCommandEvent* method to connect menu command events, the programmer must use a menu item *ID*, not a menu item *position* identifier.

The *connectCommandEvent* method is used for all types of menus. However, the programmer should be aware of the differences in coding event *handler*s for *SystemMenu* and other types of menus.

Details:

Sets the *.systemErrorCode* on error. The system error code is set this way in addition to what the OS might set:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The dialog argument was omitted and the menu does not have an assigned dialog.

ERROR_INVALID_PARAMETER (87)

The parameter is incorrect. **Meaning:** The *methodName* argument can not be the empty string.

ERROR_WINDOW_NOT_DIALOG (1420)

The window is not a valid dialog window. **Meaning:** The *dlg* argument is not a **`PlainBaseDialog`**, (or subclass of course.)

ERROR_NOT_ENOUGH_MEMORY (8)

Not enough storage is available to process this command. **Meaning:** The dialog message table is full.

Example:

This example example connects the *Open* menu item in the *File* submenu of the menu bar to the *onFileOpen* method in the dialog. Then the *Close*, *Save*, etc., menu items are connected. Since the menu bar has already been attached to the dialog, the third optional argument to *connectCommandEvent* is left out:

```
::method initDialog
  expose menuBar

  menuBar~attachTo(self)

  menuBar~connectCommandEvent(IDM_OPEN, onFileOpen)
  menuBar~connectCommandEvent(IDM_CLOSE, onFileClose)
  menuBar~connectCommandEvent(IDM_SAVE, onFileSave)
  menuBar~connectCommandEvent(IDM_SAVEAS, onFileSaveAs)
  ...
```

28.3.9. connectMenuEvent

```
>>--connectMenuEvent(--mName-,-event--+-----------+--+--------+--)------------><
```

```
                                             +-,-hFilter-+  +-,-dlg--+
```

Connects menu *event* notifications to a method in the Rexx dialog. These are menu notifications *other* than the *menu command event* notifications.

Arguments:

The arguments are:

mName [required]

The method name of the event handler to be connected. This can not be the empty string.

event [required]

Exactly one of the following keywords, case is not significant, that indicates the event to be connected:

CONTEXTMENU	INITMENU	INITMENUPOPUP

CONTEXT

Notifies a window that the user clicked the right-clicked the mouse button in the window. The application should display a context menu. The *connectContextMenu* class method or the *connectContextMenu* instance method provide the same functionality and may be more convenient to use.

INITMENU

Notifies the application that a menu is about to become active. The event occurs when the user clicks an item on the menu bar, clicks on the system menu, or presses a menu key. This notification allows the application to modify the menu before it is shown.

The notification is also generated before a context menu is about to become active, but only if the application displays a context menu. I.e., the event notification is not generated if the application uses a context menu, but does not actually display it for some reason.

INITMENUPOPUP

Notifies the application that a drop-down menu or submenu is about to become active. This allows the application to modify the menu before it is shown, without changing the entire menu.

hFilter [optional]

The Windows *handle* of an object to use to filter the connected notification. The use of this filter is specific to the event being connectd. For the CONTEXTMENU event it can be a window handle, in which case only notifications for that specific window are generated. When this menu is a menu *MenuBar*, then for the INITMENUPOPUP event it can be the menu handle of a submenu, in which case only notifications for that specific submenu are generated. When this menu is not a menu bar, the argument is ignored. For the INITMENU event, this argument is ignored.

dlg [optional]

The dialog being connected to the event. By default the dialog this menu is attached or assigned to is used. However, any dialog can be used. In most cases, it only makes sense to connect the menu notification to the dialog the menu is attached to.

Recall that menu objects exist and are not tied to dialog objects. Menu event connections can be, and maybe often are, made before a menu is assigned or attached to a dialog. Recall also that submenus in a menubar never have an owner dialog. In these cases, omitting the *dlg* argument results in *no* connection being made.

Return value:

True on success, false on error.

Remarks:

The INITMENU notification is only sent when a menu is first accessed, only one INITMENU notification is generated for each access. Moving the mouse across several menu items while holding down the button does not generate new messages. The INITMENU notification gives the programmer the opportunity to modify the entire menu before the user sees it. If the application only needs to modify a small part of the menu, the INITMENUPOPUP notification allows the programmer to only modify a specific portion of the menu and only if it is actually going to be displayed.

When this menu is a menu bar, connecting the INITMENUPOPUP event and not using the optional *hFilter* argument results in *all* INITMENUPOPUP events sent to the *underlying* dialog invoking the connected event handler. Note that this will include notifications that the programmer may not at first expect. If the user clicks on the system menu, an INITMENUPOPUP notification is generated. If the application uses a context menu, the INITMENUPOPUP notification is generated before the context menu is shown. The connected event handler will also be invoked for these notifications.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The dialog argument was omitted and the menu does not have an assigned dialog.

ERROR_NOT_ENOUGH_MEMORY (8)

Not enough storage is available to process this command. **Meaning:** The dialog message table is full.

ERROR_WINDOW_NOT_DIALOG (1420)

The window is not a valid dialog window. **Meaning:** The dialog argument was not ommitted, but the object is not a *dialog* object.

Example:

This example connects the INITMENU event to the *menuInit* method in the dialog:

```
::method initDialog
  expose menuBar

  menuBar~attachTo(self)
  menuBar~connectMenuEvent(menuInit, INITMENU)
```

28.3.9.1. Context Menu Event Handler

Note that the event handler for the CONTEXTMENU event connected through the *connectMenuEvent*, described in this section, is the same as the event handler *described* for the CONTEXTMENU event connected through one of the *PopupMenu* class methods. The event and event handler are the same no matter how the event connection is made.

The event handler for the context menu event is invoked when the user right-mouse clicks on a window, types SHIFT-F10 on the keyboard, or types the VK_APPS key on the keyboard. The VK_APPS key is the *Applications* key on a Natural keyboard.

The interpreter does not wait for the return from the event handler, so the method does not need to return a value. However, good practice would be to always return a value from an event handler. **`.true`**, the notification was processed, is a good value to return.

```
::method onContextMenu unguarded
  use arg hwnd, x, y

  return 0
```

Arguments:

The event handling method receives 3 arguments:

hwnd

The window *handle* of the window the user clicked the mouse on, or the window that has the focus if the user generated the context menu event with the keyboard. Note that this will only be the dialog window handle of the user clicks on the dialog background. Quite often it is going to be one of the dialog controls.

x

The X coordinate of the mouse position, in screen *coordinates*) coordinates, at the time of the mouse click. Or, -1 if the user generated the context menu event by using the keyboard.

y

The Y coordinate of the mouse position, in screen *coordinates*) coordinates, at the time of the mouse click. Or, -1 if the user generated the context menu event by using the keyboard.

Return:

0 makes a good return value.

Example

The following example shows the event handler method that is only invoked for a list view in the application. Note that since the application uses the *show* method to put up the context menu, the handling of the menu command item selection is done in a menu command event *handler*.

```
::method initDialog
  expose shortCutMenu
  ...

  shortCutMenu = self~createMenu
  shortCutMenu~assignTo(self)
  shortCutMenu~connectMenuEvent(onContext, 'CONTEXTMENU)

  ...

::method onContext
  expose shortCutMenu listView
  use arg hwnd, x, y

  if x == -1, y == -1 then do
    -- The keyboard was used, not the mouse.  Position the context menu as
    -- at the lower right of the list view.
    rect = listView~windowRect
    x = rect~right - .SM~cxVScroll + 15
    y = rect~bottom - 15
  end
  -- pos is the point on the screen, in pixels, to place the context menu.
  pos = .Point~new(x, y)

  -- Show the menu.
  ret = shortCutMenu~show(pos)
  if ret == -1 then do
    say "shortCutMenu~show() failed SystemErrorCode:' || -
    .SystemErrorCode SysGetErrortext(.SystemErrorCode)
  end
```

28.3.9.2. InitMenu Event Handler

The event handler for the init menu event is invoked when the user clicks on a menu or uses a menu key. The menu is about to become active, but has not actually been shown yet. This allows the programmer to modify the menu before it is displayed.

The programmer must return **`.true`** or **`.false`** from the event handler, and the interpreter *waits*) for this return.

```
::method onInitMenu unguarded
  use arg hMenu

  return .true
```

Arguments:

The event handling method receives 1 arguments

hMenu

The Windows *handle* for the menu about to become active.

Return:

Return true to indicate the notification was processed, otherwise return false. Note that the Microsoft documentation is vague in this area. Experimentation seems to indicate that, for dialogs, it does not matter if true or false is returned.

Remarks:

The INITMENU notification is only generated when a menu is first accessed and only one notification is generated for each access. For example, moving the mouse across several menu items while holding down the button does not generate new notifications.

Example

The following example shows an event handler that disables some (most) of the items in a system menu before it is shown. Whether or not the items are disabled is controlled by an instance variable that is set in other parts of the application:

```
::method onInitMenu
  expose sysMenu disableItems
  use arg hMenu

  if disableItems then do
    sysMenu~disable(SC_MAXIMIZE)
    sysMenu~disable(SC_MINIMIZE)
    sysMenu~disable(SC_MOVE)
    sysMenu~disable(SC_CLOSE)
    return .true
  end
  else do
    return .false
  end
```

28.3.9.3. InitMenuPopup Event Handler

The event handler for the init menu popup event is invoked when a submenu is about to become active. This includes the system menu or a context menu. This allows the programmer to modify a portion of a menu, rather than the whole menu.

The programmer must return **`.true`** or **`.false`** from the event handler, and the interpreter *waits*) for this return.

```
::method onInitMenuPopup unguarded
  use arg position, isSystemMenu, hMenu

  return .true
```

Arguments:

The event handling method receives 3 arguments:

position

The one-based position index of the submenu. For instance if the submenu is the third from the left in a menu bar, then position will be 3.

isSystemMenu

True if the submenu about to become active is the system menu, otherwise false

hMenu

The Windows *handle* for the submenu about to become active.

Return:

Return true to indicate the notification was processed, otherwise return false. Note that the Microsoft documentation is vague in this area. Experimentation seems to indicate that, for dialogs, it does not matter if true or false is returned.

Remarks:

If both the *INITMENU* and INITMENUPOPUP events are connected, then two successive event handlers will often be invoked. Take for instance a menu bar that contains a *File* submenu. If the user clicks on the *File* menu, the operating system first generates an INITMENU notification (for the menubar about to become active) and then generates an INITMENUPOPUP notification (for the *File* submenu about to be come active.)

Example

The following example is from an application with a large menu, with an *Edit* menu that has a *Copy* menu item. If there is nothing selected, the *Copy* menu item is disabled, if and only if, the user actually navigates to the *Edit* submenu.

```
  ...
  hMenu = menuBar~getPopup(IDM_EDIT_SUBMEN)~hMenu
  menuBar~connectMenuEvent(onEditMenuPopup, 'INITMENUPOPUP', hMenu)
  ...

::method on EditMenuPopup unguarded
  expose menuBar
  use arg pos, isSysMenu, hMenu

  if self~haveSomeSelection then menuBar~enable(IDM_COPY)
  else menuBar~disable(IDM_COPY)
```

28.3.10. connectSomeCommandEvents

```
>>--connectSomeCommandEvents(--ids-+-------+-+-------------+-+-------+--)-----><
                                   +-,-mth-+ +-,-byPosition-+ +-,-dlg-+
```

Connects a collection of menu command items to a method, or methods, in the Rexx dialog.

Arguments:

The arguments are:

ids [required]

A **`Collection`** object containing the menu item *ID*s to connect. The IDs can be by position IDs or resource IDs, depending on the value of the *byPosition* argument. However, they must be all the same type, all resource IDs or all by position IDs.

mth [optional]

Connect all menu command items to the method of this name. The default is to connect all menu command items to a method name composed from the text of the command item. If not omitted, *mth* can not be the empty string.

byPosition [optional]

If true, the IDS are by positional IDs, otherwise the are resource IDs. The default is false, they are resource IDs.

dlg [optional]

Connect the command items to the method(s) of this dialog object. The default is to connect the command items to the owner dialog of this menu.

Return value:

Returns true on success, false on error.

Remarks:

The *ids* argument can be any ooRexx collection with a *makeArray* method that returns an array whose *items* are the menu item IDs to connnect.

This method quits when an error is detected. This implies that on an error return some menu items may have been connected.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The dialog argument was omitted and the menu does not have an assigned dialog.

ERROR_NOT_ENOUGH_MEMORY (8)

Not enough storage is available to process this command. **Meaning:** The dialog message table is full.

ERROR_INVALID_PARAMETER (87)

The parameter is incorrect. **Meaning:** One or more of the specified item IDs is not a menu command item. Or the msg argument was used, but it is the empty string.

ERROR_WINDOW_NOT_DIALOG (1420)

The window is not a valid dialog window. **Meaning:** The dialog argument was not ommitted, but the *dlg* is not a dialog *dialog* object.

Example:

This example connects four menu command items in the menu bar, by their symbolic IDs, to 4 methods in the dialog. The method names will be constructed from the text of the menu item:

```
::method enuConnectItems
  expose menuBar
  use strict arg

  ids = .array~of(IDM_EDIT_COPY, IDM_EDIT_PASTE, IDM_EDIT_CUT, IDM_EDIT_DELETE)

  if \ menuBar~connectSomeCommandEvents(ids) then self~error('Connect Some Menu Items
 Failed')
```

28.3.11. deletePopup

```
>>--deletePopup(--id--+---------------+--)-------><
                      +-,-byPosition--+
```

Deletes the specified popup menu from this menu.

Arguments:

The arguments are:

id [required]

The *ID* of the popup menu to be deleted. This can be a position ID or a resource ID, depending on the value of *byPosition*.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

Returns true on success, false on error.

Remarks:

When a popup menu (a submenu) is deleted, the operating system frees all resources used by the popup menu and any references to the popup are no longer valid. To be explicit, this means that any *PopupMenu* objects that represent this submenu are no longer valid, which would also include any menu that has this submenu as one of its submenus.

To remove a submenu from a menu, and still use the submenu, use the *removePopup* method.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error code may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The specified menu item is not a popup menu.

Example:

This example removes the *View* popup menu from a menubar and deletes it, freeing up the system resources used for it:

```
menuBar~deletePopup(IDM_VIEW)
```

28.3.12. destroy

```
>>--destroy--------------------------------------><
```

Releases the operating system resources for the *underlying* menu of this **Menu** object. Menubars attached to a dialog, or submenus that are a part of a menubar, should not be destroyed. A menubar attached to a window is automatically destroyed by the operating system when the window is destroyed. When the operating system destroys a menubar or popup menu, it does so recursively. All submenus of the menu are also destroyed.

Arguments:

This method has no arguments.

Return value:

True on success, otherwise false.

Remarks:

It is only when a menubar is not attached to a window, or a popup menu is not part of a menubar, that a menu needs to be destroyed. This should be done before the application ends, but it could be done at any time if the menu is no longer needed. Do not destroy a submenu of a menubar unless the submenu has been *removePopup* from the menubar, when the menubar is attached to a dialog. The results are unpredictable.

When the *destroy* method is invoked, if this menu is a menubar attached to a dialog, the menubar is first *detach*. The menubar is then destroyed. This in turn will cause the dialog to redraw its client area and all controls in the dialog will be moved up by the amount of space the menubar had occupied.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example comes from an application that has a simple mode and an *advanced* mode. The simple and advanced modes each have their own menubars. When the user switches modes, the existing menubar is detached and the menubar for the new mode is attached to the dialog. When the dialog is ended, the menubar not currently in use is destroyed:

```
::method leaving
  expose advancedMenubar simpleMenubar

  if self~isAdvancedMode then simpleMenubar~destroy
  else advancedMenubar~destroy
```

28.3.13. disable

```
>>--disable(--IDs--+--------------+--)----------><
                   +-,-byPosition-+
```

Disables one or more menu items. Separators can not be disabled. Note that disable and gray are the exact same thing.

Arguments:

The arguments are:

IDs [required]

The menu item *ID(s)* to be disabled, may be *symbolic* or numeric. This can be a single ID or a collection of IDs. The ID(s) can be position IDs or resource IDs, depending on the value of *byPosition*. However, if this is a collection of IDs there must be all the same type, all resource IDs or all position IDs.

byPosition [optional]

If true, the item IDs are positional IDs, otherwise they are resource IDs. The default is false, resource IDs.

Return value:

True on success, otherwise false.

Remarks:

To disable more than 1 menu item at a time, use a collection object, such as an **`array`**, where each item in the collection is the menu item ID of the item to be disabled.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The menu item specified to be disabled is a separator.

Example:

This example comes from an application that only allows the user to open 1 file at a time. When the current file is closed, the Close File menu item is disabled and the Open File menu item is enabled.

```
::method onCloseFile unguarded
  expose menuBar

  menuBar~disable(IDM_FILE_CLOSE)
  menuBar~enable(IDM_FILE_OPEN)

  self~closeCurrentFile
  return 0
```

28.3.14. enable

```
>>--enable(--IDs--+--------------+--)-----------><
                  +-,-byPosition-+
```

Enables one or more menu items. Separators can not be enabled.

Arguments:

The arguments are:

IDs [required]

The menu item *ID(s)* to be enabled, may be *symbolic* or numeric. This can be a single ID or a collection of IDs. The ID(s) can be position IDs or resource IDs, depending on the value of *byPosition*. However, if this is a collection of IDs there must be all the same type, all resource IDs or all position IDs.

byPosition [optional]

If true, the item IDs are positional IDs, otherwise they are resource IDs. The default is false, resource IDs.

Return value:

True on success, otherwise false.

Remarks:

To enable more than 1 menu item at a time, use a collection object, such as an **`array`**, where each item in the collection is the menu item ID of the item to be enabled.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The menu item specified to be enabled is a separator.

Example:

This example comes from an application that only allows the user to open 1 file at a time. When the current file is opened, the Open File menu item is disabled and the Close File menu item is enabled.

```
::method onOpenFile unguarded
  expose menuBar

  if self~openFile then do
    menuBar~disable(IDM_FILE_OPEN)
    menuBar~enable(IDM_FILE_CLOSE)
  end

  return 0
```

28.3.15. getAutoConnectStatus

```
>>--getAutoConnectStatus-------------------------><
```

Returns a **`Directory`** object containing the current status of *autoconnection*.

Arguments:

This method has no arguments.

Return value:

Returns a **`.Directory`** object with the following indexes:

AUTOCONNECT
: If true, autoconnection is on. If false, autoconnection is off

METHODNAME
: The method name for autoconnection, if set. The **`.nil`** object if no method name is set, or autoconnection is off.

Remarks:

Use the *setAutoConnection* method to change the status of autoconnection.

Example:

This code could be used to determine if autoconnection is on or off:

```
if menu~getAutoConnectStatus~autoConnect then do
  -- autoconnection is on, do something ...
end
else do
  -- autoconnection is no on, do something else ...
end
```

28.3.16. getCount

```
>>--getCount-----------------------------------------><
```

Determines the number of menu items in this menu.

Arguments:

There are no arguments.

Return value:

The number of menu items in the menu, or -1 on error.

Remarks:

The number of items in a menu will include the separator and submenu items, but not the count of items in the submenus. In other words, it is not a recursive count, just a count of the single menu.

Details:

Sets the *.systemErrorCode*.

Example:

```
count = menuObj~count
if count \== -1 then do
  say 'The menu has' count 'items.'
end
else do
  say 'Error getting count of menu items:'
  say '  Error Code:' .systemErrorCode SysGetErrortext(.systemErrorCode)
end
```

28.3.17. getHandle

```
>>--getHandle-----------------------------------><
```

Retrieves the Windows *handle* of the *underlying* menu this Rexx object represents.

Arguments:

This method has no arguments.

Return value:

The handle to the menu.

Remarks:

If a menu is released through the *destroy*() method, the menu handle will then be null and is no longer valid. The menu handle is a Rexx **`Pointer`** object and therefore has the methods of that class. Specifically it has the *isNull*() method, which can be used to test if the handle is still valid.

The value returned from *getHandle* is identical to the value of the *hMenu* attribute.

Example:

This example gets the handle for an Edit submenu and then uses it as a filter in the *connectMenuEvent* method:

```
...

editMenu = menuBar~getPopup(IDM_POP_EDIT)
hEditMenu = editMenu~getHandle

menuBar~connectMenuEvent(onEditMenuPopup, 'INITMENUPOPUP', hEditMenu)
```

28.3.18. getHelpID

```
>>--getHelpID------------------------------------><
```

Gets the Help context identifier for this menu.

Arguments:

There are no arguments for this method.

Return value:

On success, returns the help ID, or 0 if the help ID for the menu is not set. On failure, returns -1.

Details:

Sets the *.systemErrorCode*.

28.3.19. getID

```
>>--getID(--pos--)-------------------------------><
```

Gets the resource ID of the specified menu item. This is always done by position, if the ID is already known, there is no use querying for it.

Arguments:

The single argument is:

pos

The *menu item position* identifier whose resource ID is required.

Return value:

Returns the resource ID of the specified menu item, which is always greater than 0, or -1 on error.

Remarks:

Recall that when using menu item position identifiers, the item must be in the immediate menu, the menu that actually holds the menu item. The item can not be in a submenu of the menu.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example searches a menu until it finds a menu item that matches the specified text. If it finds that item, it returns the item's resource ID, otherwise it returns -1.

```
::method retrieveID private
  use strict arg menu, label

  do i = 1 to menu~getCount
    if menu~getText(i, .true) == label then do
      id = menu~getID(i)
      if id > 0 then return id
    end
  end

  return -1
```

28.3.20. getItemState

```
>>--getItemState(--id--+---------------+--)------><
                       +-,-byPosition--+
```

Returns a string of 0 or more keywords that indicate the current state of the specified menu item.

Arguments:

The arguments are:

id [required]

The *ID* of the menu item to get the state of. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

Returns -1 on error, otherwise a text string is returned that can contain zero or more of the following keywords, separated by blanks, which indicate the menu item's state:

CHECKED

The menu item is checked.

DEFAULT

The menu item is the default menu item.

DISABLED GRAYED
: The menu item is disabled. Disabled and grayed are exactly the same thing. If the menu item is disabled, the returned keyword string will always contain *DISABLED GRAYED*.

ENABLED
: The menu item is not disabled

HILITE
: The menu item is high lighted, that is, it is drawn with the selected appearance.

UNCHECKED
: The menu item is not checked

UNHILITE
: The menu item is not high lighted, that is, it is not drawn with the selected state.

Remarks:

If the menu item is a separator, the keyword string will be the empty string.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

28.3.21. getMaxHeight

```
>>--getMaxHeight---------------------------------><
```

Gets the current maximum height for this menu, in pixels. Note that 0, indicates that the maximum height is the height of the screen.

Arguments:

There are no arguments.

Return value:

The maximum height of this menu in pixels, or -1 on error. As mentioned, a return of 0 indicates that the maximum height is the height of the screen.

Remarks:

By default, the maximum height of a menu is the height of the screen. This can be changed by the programmer through the *setMaxHeight*.

Details:

Sets the *.systemErrorCode*.

Example:

This example gets the current maximum height of the File drop down menu and checks if it is equal to the height of the dialog. If not, it changes the maximum height to the height of the dialog:

```
::method checkHeight private
  expose menuBar

  size = self~getRealSize

  filePopup = menuBar~getPopup(IDM_FILE)
  if filePopup~getMaxHeight <> size~height then filePopup~setMaxHeight(size~height)

  return 0
```

28.3.22. getMenuHandle

```
>>--getMenuHandle(--id--+---------------+--)----->< 
                        +-,-byPosition--+
```

Retrieves the Windows *handle* of the *underlying* menu for the specified popup menu of this menu.

Arguments:

The arguments are:

id [required]

The *ID* of the popup menu to get the handle for. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

Returns the handle for the submenu on success, or 0 on failure.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error code may be set by ooDialog:

ERROR_INVALID_PARAMETER (87)

The parameter is incorrect. **Meaning:** the item specified is not a popup menu of this menu.

Sets the *.systemErrorCode*.

Example:

This example gets the handle of a submenu in the menubar in order to use it as a filter in the *connectMenuEvent* method:

```
  ...

  editMenu = menuBar~getMenuHandle(IDM_POP_EDIT)

  menuBar~connectMenuEvent(onEditMenuPopup, 'INITMENUPOPUP', hEditMenu)
```

28.3.23. getPopup

```
>>--getPopup(--id--+---------------+--)----------><
                   +-,-byPosition--+
```

Gets the specified popup menu from this menu. This menu is unchanged.

Arguments:

The arguments are:

id [required]

The *ID* of the popup menu to get the handle for. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

The specified **`PopupMenu`** object on success, or 0 on failure.

Remarks:

The popup menu returned by the *getPopup* remains a submenu of the parent menu it was gotten from. Therefore the programmer should not invoke the *destroy* or *releaseMenuHandle* methods on the returned object. The underlying submenu will automatically be destroyed by the operating system when its parent menu is destroyed.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:
ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The specified menu item is not a popup menu.

OR_INVALID_OID (1911)

The object specified was not found. **Meaning:** Internally, some error occurred in trying to get the Rexx **`Popupmenu`** object that represents the underlying submenu.

Sets the *.systemErrorCode*.

Example:

This example gets a popup menu from a menubar so that it can be used as a context menu. Note that the popup menu is gotten by its position ID rather than its resource ID:

```
...
dlgPopup = menubar~getPopup(1, .true)
...
```

28.3.24. getText

```
>>--getText(--id--+---------------+--)-----------><
                  +-,-byPosition--+
```

Gets the text, the label, for the specified menu item. Separators do not have any text.

Arguments:

The arguments are:

id [required]

The *ID* of the menu item to get the text for. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

The text of the specified menu item on success, the empty string on failure.

Remarks:

The label for a menu item can contain an ampersand symbol, which is not visible to the user. The text returned is the complete text set for the menu item.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:
ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The menu item specified is a separator.

Example:

This example comes from some debug code. The programmer has connected all menu command items (*connectAllCommandEvents*), but is having trouble supplying the correct event handling method in the code. The debug code gets the text of the menu item and then uses *itemTextToMethodName* to determine the correct method name:

```
::method doDebug private
  expose menuBar
  use strict arg id

  text = menuBar~getText(id)
  say 'Method name for menu item' id 'should be:' menuBar~itemTextToMethodName(text)
```

28.3.25. getType

```
>>--getType(--id--+---------------+--)-----------><
                  +-,-byPosition--+
```

Returns a string of 0 or more keywords that indicate the type of the specified menu item.

Arguments:

The arguments are:

id [required]

The *ID* of the menu item to get the text for. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

Returns -1 on error, or a text string that can contain zero or more of the following keywords, separated by blanks, which indicate the menu item's type:

BITMAP

The menu item is displayed using a bitmap.

COMMAND

The menu item is not a submenu or a separator, that is, the user will expect a command to be carried out when clicking on, or selecting, this item.

MENUBARBREAK
: The menu item is placed on a new line (for a menubar) or in a new column (for a drop-down menu, submenu, or shortcut menu). For a drop-down menu, submenu, or shortcut menu, a vertical line separates the new column from the old.

MENUBREAK
: The menu item is placed on a new line (for a menu bar) or in a new column (for a drop-down menu, submenu, or shortcut menu). For a drop-down menu, submenu, or shortcut menu, the columns are not separated by a vertical line.

OWNERDRAW
: The application is responsible for drawing the menu item. ooDialog does not currently support this type, the keyword is documented for completeness.

RADIO
: If selected, the menu item is displayed using a radio-button mark instead of a check mark.

RIGHTJUSTIFY
: The menu item and any subsequent items are right-justified. This type is only used in a menubar.

RIGHTORDER
: Specifies that menus cascade right-to-left (the default is left-to-right). This is used to support right-to-left languages, such as Arabic and Hebrew.

SEPARATOR
: The menu item is a separator.

STRING
: The menu item is displayed using a text string

SUBMENU
: The menu item is a submenu, that is a popup menu.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

28.3.26. gray

```
>>--gray(--IDs--+--------------+--)--------------><
                +-,-byPosition-+
```

Grays one or more menu items. Separators can not be grayed. Note that disable and gray are the exact same thing.

Arguments:

The arguments are:

IDs [required]
: The menu item *ID(s)* to be grayed, may be *symbolic* or numeric. This can be a single ID or a collection of IDs. The ID(s) can be position IDs or resource IDs, depending on the value of *byPosition*. However, if this is a collection of IDs there must be all the same type, all resource IDs or all position IDs.

byPosition [optional]

If true, the item IDs are positional IDs, otherwise they are resource IDs. The default is false, resource IDs.

Return value:

True on success, otherwise false.

Remarks:

To gray more than 1 menu item at a time, use a collection object, such as an **`array`**, where each item in the collection is the menu item ID of the item to be grayed.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:
ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The menu item specified to be grayed is a separator.

Example:

This example comes from an application that only allows the user to open 1 file at a time. When the current file is closed, the Close File menu item is grayed and the Open File menu item is enabled.

```
::method onCloseFile unguarded
  expose menuBar

  menuBar~gray(IDM_FILE_CLOSE)
  menuBar~enable(IDM_FILE_OPEN)

  self~closeCurrentFile
  return 0
```

28.3.27. hilite

```
>>--hilite(--IDs--+--------------+--)------------><
                  +-,-byPosition-+
```

High lights one or more menu items. Separators can not be high lighted. A high lighted menu item is drawn as though it were the selected item.

Arguments:

The arguments are:

IDs [required]

The menu item *ID(s)* to be high lighted, may be *symbolic* or numeric. This can be a single ID or a collection of IDs. The ID(s) can be position IDs or resource IDs, depending on the value of *byPosition*. However, if this is a collection of IDs there must be all the same type, all resource IDs or all position IDs.

byPosition [optional]

If true, the item IDs are positional IDs, otherwise they are resource IDs. The default is false, resource IDs.

Return value:

True on success, otherwise false.

Remarks:

To high light more than 1 menu item at a time, use a collection object, such as an **`array`**, where each item in the collection is the menu item ID of the item to be high lighted.

There is not much use for this method, in the author's opinion, but it is documented for completeness since the method does exist. High lighting a menu item does not actually select the item, it just makes it look like it is selected. This would be confusing to the user. In addition, although this method will accept a collection of item IDs, as do the other similar methods, it makes little sense to high light more than one menu item at a time.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The menu item specified to be high lighted is a separator.

28.3.28. insertItem

```
>>--insertItem(-idBefore-,-id-,-text--+----------+--+--------+------->
                                      +-,-state--+  +-,-type-+

>------------+-------------+--+----------+--+---------+--)---------><
             +-,-byPosition-+  +-,-connect-+  +-,-mName-+
```

Inserts a new menu command item into this menu.

Arguments:

The arguments are:

idBefore [required]

The newly inserted item is inserted before the item with this *ID*. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

id [required]

The *ID* of the menu item to be inserted. May be *symbolic* or numeric.

text [required]

The text, the label, for the new menu item.

state [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. These specify the state of the new item. If this argument is omitted or the empty string the state is set to ENABLED UNHILITE NOTDEFAULT UNCHECKED.:

CHECKED	ENABLED	NOTDEFAULT
DEFAULT	GRAYED	UNCHECKED
DISABLED	HILITE	UNHILITE

CHECKED

The menu item is checked.

DEFAULT
: The menu item is the default menu item. Only one item in any submenu can be the default item.

DISABLED
: The menu item is disabled. Disabled and grayed are exactly the same thing.

ENABLED
: The menu item is not disabled. This keyword does not need to be specified. If neither DISABLED nor GRAYED are specified, the menu item is automatically enabled

GRAYED
: The menu item is disabled. Disabled and grayed are exactly the same thing.

HILITE
: The menu item is high lighted, that is, it is drawn with the selected appearance.

NOTDEFAULT
: The menu item is not the default menu item. This keyword does not need to be specified. If DEFAULT is not specified, the menu item is automatically not the default menu item.

UNCHECKED
: The menu item is not checked. This keyword does not need to be specified. If CHECKED is not specified, the new item will automatically be unchecked.

UNHILITE
: The menu item is not high lighted, that is, it is not drawn with the selected state. This keyword does not need to be specified, if the HILITE keyword is omitted, the NOHILITE state is set automatically.

type [optional]
: A list of 0 or more of the following keywords separated by spaces, case is not significant. These specify the type of the new item. If this argument is omitted, or the empty string, then no special type is set:

RADIO	MENUBARBREAK
RIGHTJUSTIFY	MENUBAR

RADIO
: Displays checked menu items using a radio-button mark instead of a check mark.

RIGHTJUSTIFY
: Right-justifies the menu item and any subsequent items. This value is valid only if the menu item is in a menu bar. It has no effect if the item is in a popup menu.

MENUBARBREAK
: Places the menu item on a new line (for a menu bar) or in a new column (for a drop-down menu, submenu, or shortcut menu). For a drop-down menu, submenu, or shortcut menu, a vertical line separates the new column from the old.

MENUBREAK
: This is exactly the same as MENUBARBREAK, except that no vertical line is drawn between the columns when the item is not in a menu bar.

byPosition [optional]
: If **`.true`**, *idBefore* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

connect
: Whether to automatically connect the command item to a method in the owning dialog. This is a *connection request* type of connection type. Do not mix this with *autoconnection*. The default is false. If connect is true and the menu is already attached to a dialog the connection is made immediately. Otherwise the request is put in to a queue and the connection is made when the menu is first attached to a dialog.

mName
: If *connect* is true, this can be the name of the method to connect the command event to. If *connect* is false, this argument is ignored. If *connect* is true and this argument is omitted, then the method name is automatically constructed from the text of the new item.

Return value:

Returns true on success, otherwise false.

Remarks:

If *autoconnection* is on and *connect* is set to true by the programmer then the new menu item is connected by *autoconnection* and also connected because of the *connect* request. The result of this is unpredictable. The *connect* argument should never be set to true if autoconnection is on for the menu.

When the menu command item is inserted into the menu, if there are no other items in the menu there is no ID for the *before* argument. In this case the programmer can use any positive number and the operating system will accept it. The ooDialog examples commonly use the same ID for both the *idBefore* and *id* arguments for this case. 1 is also a good choice for the *idBefore* argument.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example inserts a new menu command item into a menu bar with the resource ID of IDM_LV_PROFESSION. By position is false, (it could have been omitted,) so IDM_LV_RESTORE is a resource ID. The new item is inserted before the IDM_LV_RESTORE menu item, where ever it is in the menu bar. The *connect* argument is true, so the new menu command item is connected to the *onOrderByProfession* method, either immediately if the menubar is already attached to a dialog, or when the menubar is first attached to a dialog:

```
  menuBar~insertItem(IDM_LV_RESTORE, IDM_LV_PROFESSION, "Order by Profession", -
                     'DEFAULT CHECKED', 'RADIO', .false, .true, onOrderByProfession)
```

28.3.29. insertPopup

```
>>--insertPopup(-idBefore-,-id-,-pop-,-txt-+----------+-+--------+-+---------+--)--><
                                           +-,-state--+ +-,-type-+ +-,-byPos-+
```

Inserts a new submenu into this menu.

Arguments:

The arguments are:

idBefore [required]

The newly inserted submenu is inserted before the item with this *ID*. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPos*.

id [required]

The resource *ID* of the submenu to be inserted. May be *symbolic* or numeric.

pop [required]

The *PopupMenu* object being inserted.

txt [required]

The text, the label, for the new submenu.

state [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. These specify the state of the new submenu. If this argument is omitted or the empty string the state is set to ENABLED UNHILITE NOTDEFAULT:

DEFAULT	GRAYED	UNHILITE
DISABLED	HILITE	
ENABLED	NOTDEFAULT	

DEFAULT

The submenu is the default menu item. Only one item in any menu can be the default item.

DISABLED

The summenu is disabled. Disabled and grayed are exactly the same thing.

ENABLED

The submenu is not disabled. This keyword does not need to be specified. If neither DISABLED nor GRAYED are specified, the submenu is automatically enabled

GRAYED

The submenu is disabled. Disabled and grayed are exactly the same thing.

HILITE

The submenu is high lighted, that is, it is drawn with the selected appearance.

NOTDEFAULT

The submenu is not the default menu item. This keyword does not need to be specified. If DEFAULT is not specified, the submenu is automatically not the default menu item.

UNHILITE

The submenu is not high lighted, that is, it is not drawn with the selected state. This keyword does not need to be specified, if the HILITE keyword is omitted, the NOHILITE state is set automatically.

type [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. These specify the type of the new submenu. If this argument is omitted or the empty string then no special type is set:

RIGHTJUSTIFY	MENUBARBREAK
RIGHTORDER	MENUBAR

RIGHTJUSTIFY
: Right-justifies the submenu and any subsequent items. This value is valid only if the submenu is in a menu bar. It has no effect if the item is in a popup menu.

RIGHTORDER
: Specifies that menus cascade right-to-left (the default is left-to-right). This is used to support right-to-left languages, such as Arabic and Hebrew.

MENUBARBREAK
: Places the submenu on a new line (for a menu bar) or in a new column (for a drop-down menu, submenu, or shortcut menu). For a drop-down menu, submenu, or shortcut menu, a vertical line separates the new column from the old.

MENUBREAK
: This is exactly the same as MENUBARBREAK, except that no vertical line is drawn between the columns when the submenu is not in a menu bar.

byPosition [optional]
: If **`.true`**, *idBefore* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

True on success, false on error.

Remarks:

When the submenu is inserted into the menu, if there are no other items in the menu there is no ID for the *before* argument. In this case the programmer can use any positive number and the operating system will accept it. The ooDialog examples commonly use the same ID for both the *idBefore* and *id* arguments for this case. 1 is also a good choice for the *idBefore* argument.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example shows the start of the creation of a binary menubar. First a new empty menubar is instantiated. Then a new popup menu is instantiated with the resource id of IDM_HELP. One menu item is inserted into the popup menu, and the popup menu is inserted into the menubar. Note that the menubar at this point does not have any menu items, so there is no menu item for the *idBefore* argument. In this case the operating system does not seem to care what number is used for the *idBefore* argument. The examples in ooDialog generally will use the same ID as the ID of the menu item being inserted:

```
menu = .BinaryMenuBar~new(.nil, IDM_MENUBAR, , self)

subMenu = .PopupMenu~new(IDM_HELP))
subMenu~insertItem(IDM_ITEM_ABOUT, IDM_ITEM_ABOUT, "About")

menu~insertPopup(IDM_HELP, IDM_HELP, subMenu, "Help", , 'RIGHTJUSTIFY')
```

28.3.30. insertSeparator

```
>>--insertSeparator(--idBefore-,-id--+---------+--)-------------><
                                     +-,-byPos-+
```

Inserts a separator into this menu at the specified position.

Arguments:

The arguments are:

idBefore [required]

The new separator is inserted before the item with this *ID*. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPos*.

id [required]

The resource *ID* of the separator to be inserted. May be *symbolic* or numeric.

byPosition [optional]

If **.true**, *idBefore* is a positional ID, otherwise it is a resource ID. The default is **.false**.

Return value:

Returns true on success, false on error.

Remarks:

When the separator is inserted into the menu, if there are no other items in the menu there is no ID for the *before* argument. In this case the programmer can use any positive number and the operating system will accept it. The ooDialog examples commonly use the same ID for both the *idBefore* and *id* arguments for this case. 1 is also a good choice for the *idBefore* argument.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example creates a new popup menu and inserts several menu items, including a separator:

```
subMenu = .PopupMenu~new(IDM_POP_FILE)

subMenu~insertItem(IDOK, IDOK, "Exit")
subMenu~insertItem(IDOK, IDM_MI_LEAVE, "Leave")
subMenu~insertItem(IDM_MI_LEAVE, IDM_MI_TEST_ITEM, "Test Item")
subMenu~insertItem(IDM_MI_LEAVE, IDM_MI_CONTEXT, "Context")
subMenu~insertSeparator(IDM_MI_LEAVE, IDM_SEPARATOR)
```

28.3.31. isChecked

```
>>--isChecked(--id--+---------------+--)---------><
                    +-,-byPosition--+
```

Determines if the specified menu item is checked.

Arguments:

The arguments are:

id [required]

The *ID* of the menu item to determine the checked status of. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

byPosition [optional]

If **.true**, *id* is a positional ID, otherwise it is a resource ID. The default is **.false**.

Return value:

Returns true if the menu item is checked, othewise false.

Remarks:

This method raises a syntax condition on any failure, since there is no way to indicate failure in the return. The syntax condition will indicate why the method failed. If the failure was a Win32 API failure, the error text will contain the value of the *.systemErrorCode* so that the programmer can determine the cause of the failure. However, it is very unlikely that the Win32 API will fail. The more common cause of a failure would be the use of an invalid symbolic ID.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

28.3.32. isCommandItem

```
>>--isCommandItem(--id--+---------------+--)-----><
                        +-,-byPosition--+
```

Determines if the specified menu item is command menu item, that is if the menu item is one that a user would expect to carry out a command.

Arguments:

The arguments are:

id [required]

The *ID* of the menu item to determine if it is a command item. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

Returns true if the menu item is a command menu item, othewise false.

Remarks:

This method raises a syntax condition on any failure, since there is no way to indicate failure in the return. The syntax condition will indicate why the method failed. If the failure was a Win32 API failure, the error text will contain the value of the *.systemErrorCode* so that the programmer can determine the cause of the failure. However, it is very unlikely that the Win32 API will fail. The more common cause of a failure would be the use of an invalid symbolic ID.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

28.3.33. isDisabled

```
>>--isDisabled(--id--+---------------+--)--------><
                     +-,-byPosition--+
```

Determines if the specified menu item is disabled or not.

Arguments:

The arguments are:

id [required]

The *ID* of the menu item to determine if it is disabled. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

Returns true if the menu item is disabled, othewise false.

Remarks:

This method raises a syntax condition on any failure, since there is no way to indicate failure in the return. The syntax condition will indicate why the method failed. If the failure was a Win32 API failure, the error text will contain the value of the *.systemErrorCode* so that the programmer can determine the cause of the failure. However, it is very unlikely that the Win32 API will fail. The more common cause of a failure would be the use of an invalid symbolic ID.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

28.3.34. isEnabled

```
>>--isEnabled(--id--+---------------+--)---------><
                    +-,-byPosition--+
```

Determines if the specified menu item is enabled or not.

Arguments:

The arguments are:

id [required]

The *ID* of the menu item to determine if it is enabled. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

Returns true if the menu item is enabled, othewise false.

Remarks:

This method raises a syntax condition on any failure, since there is no way to indicate failure in the return. The syntax condition will indicate why the method failed. If the failure was a Win32 API failure, the error text will contain the value of the *.systemErrorCode* so that the programmer can determine the cause of the failure. However, it is very unlikely that the Win32 API will fail. The more common cause of a failure would be the use of an invalid symbolic ID.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

28.3.35. isGrayed

```
>>--isEnabled(--id--+---------------+--)---------><
                    +-,-byPosition--+
```

Determines if the specified menu item is grayed or not.

Arguments:

The arguments are:

id [required]

The *ID* of the menu item to determine if it is grayed. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

Returns true if the menu item is grayed, othewise false.

Remarks:

This method raises a syntax condition on any failure, since there is no way to indicate failure in the return. The syntax condition will indicate why the method failed. If the failure was a Win32 API failure, the error text will contain the value of the *.systemErrorCode* so that the programmer can determine the cause of the failure. However, it is very unlikely that the Win32 API will fail. The more common cause of a failure would be the use of an invalid symbolic ID.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

28.3.36. isPopup

```
>>--isPopup(--id--+---------------+--)-----------><
                  +-,-byPosition--+
```

Determines if the specified menu item is a submenu (a pop up menu.)

Arguments:

The arguments are:

id [required]

The *ID* of the menu item to be queried. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

Returns **`.true`** if the menu item is a pop up menu item, otherwise **`.false`** .

Remarks:

This method raises a syntax condition on any failure, since there is no way to indicate failure in the return. The syntax condition will indicate why the method failed. If the failure was a Win32 API failure, the error text will contain the value of the *.systemErrorCode* so that the programmer can determine the cause of the failure. However, it is very unlikely that the Win32 API will fail. The more common cause of a failure would be the use of an invalid symbolic ID.

Details:

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect arguments are detected.

28.3.37. isSeparator

```
>>--isSeparator(--id--+---------------+--)-------><
                      +-,-byPosition--+
```

Determines if the specified menu item is a separator.

Arguments:

The arguments are:

id [required]

The *ID* of the menu item to be queried. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

Returns **`.true`** if the menu item is a separator menu item, otherwise **`.false`** .

Remarks:

This method raises a syntax condition on any failure, since there is no way to indicate failure in the return. The syntax condition will indicate why the method failed. If the failure was a Win32 API failure, the error text will contain the value of the *.systemErrorCode* so that the programmer can determine the cause of the failure. However, it is very unlikely that the Win32 API will fail. The more common cause of a failure would be the use of an invalid symbolic ID.

Details:

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect arguments are detected.

28.3.38. isValidItemID

```
>>--isValidItemID(--id--+---------------+--)---------><
                        +-,-byPosition--+
```

Determines if a specific menu item *ID* is valid for this menu and if one of the *is** methods, *isChecked*, *isPopup*, etc., is likely to fail.

Arguments:

The arguments are:

id [required]

The menu item ID to check.

byPosition [optional]

If the item ID is a resource ID or a positional ID. The default is false, a resource ID. Use true if the ID is a positional ID.

Return value:

True if the id is valid for the menu and if one of the *is** methods is likely to succeed, otherwise false.

Remarks:

In most of the methods for the Menu classes, if an invalid item ID is specified, the method will fail and the *.systemErrorCode* is set to the failure reason. However, in some methods there is no way to indicate that the method failed. For those methods, a syntax error is raised if the item ID is not valid.

The *isValidItemID* method can be used by the programmer to check if an item ID will raise a syntax error. The method also checks if the Win32 API used for a method succeeds. This allows the *defensive* programmer a way to check if a condition is going to be raised by invoking one of the *is** methods before invoking it.

However, a failure of the Win32 API is extremely unlikely and using an invalid menu id is easily fixed. Indeed, letting a syntax condition be raised for an invalid menu id is usually the best practice. The syntax condition will point out the incorrect id early in the development of an ooDialog program. Once fixed, the error will never happen again.

Details:

Sets the *.systemErrorCode*.

Example:

This example shows a method implemented by an overly-paranoid programmer as a replacement method for the *isSeparator* method.

```
::method isMenuSeparator private
  use strict arg menu, id, byPosition

  if menu~isValidItemID(id) then do
    if menu~isSeparator(id, byPosition) then return .true
    else return .false
  end

  -- Not a valid item id
  msg =   'The menu id:' id 'with byPosition value:' byPosition 'is ' || .endOfLine
    -
          'not valid.'                                          ||
 endOfLine~copies(2) -
          'Continue?'

  title = 'Invalid Menu Item ID Discovered'

  ret = MessageDialog(msg, self~hwnd, title, 'YESNO', 'ERROR', 'DEFBUTTON2')
  if ret == self~IDNO then do
    self~cancel:super
    return .false
  end
  return .false
```

28.3.39. isValidMenu

```
>>--isValidMenu--------------------------------------><
```

Checks if this menu is valid. This menu would not be valid if it has been *destroy* or *releaseMenuHandle*. In addition, the menu could no longer be valid for some operating system reason, although this is highly unlikely.

Arguments:

The method has no arguments.

Return value:

Returns true if the menu is valid, otherwise false.

28.3.40. isValidMenuHandle

```
>>--isValidMenuHandle(--handle--)--------------------><
```

Given a *handle*, determines if the handle is a menu handle and, if it is, if the menu is still valid.

Arguments:

The single argument is:

handle

The handle to check if it is a valid menu handle.

Return value:

True if *handle* is the handle of a valid menu, otherwise false.

28.3.41. itemTextToMethodName

```
>>--itemTextToMethodName(--text--)---------------><
```

Converts *text* to a method name as if *text* were the text of a menu command item. This method can be used as a debugging helper if the programmer has difficulty in determining what method name the ooDialog framework is constructing when automatically *connecting*/> menu command items.

The ooDialog framework constructs a method name from a string in this way: All space, tab, ampersand, colon, and plus characters are removed from the string. In addition any trailing periods are remove from the string. The remaining string is used as the method name.

Arguments:

The single argument is:

text [required]

The text string to be converted to a method name. This can not be the empty string.

Return value:

The method name the ooDialog framework would construct from *text*.

Remarks:

In the menu classes there are a number of methods that connect *menu command event*s to a method in the Rexx dialog. Most of these methods will automatically construct the connected method name if the programmer does not specify a method name. The name is constructed based on the text of the menu command item that is being connected.

Sometimes, it may be difficult for the programmer to know what the constructed name is going to be. For instance, with a compiled binary menu, the programmer might need to look at the original resource script. Or, the programmer may not recall exactly what the rules are for constructing the name. This method can help in those situations by returning the method name that would be constructed. The programmer could add some quick debug code that printed the name to the screen.

Details:

Raises syntax errors when incorrect usage is detected.

Example:

This example shows a debugging method that could be temporarily added to a dialog with a menu to determine what method name would be constructed for any menu command item:

```
::method printMethodName private
  use strict arg menu, itemID

  text = menu~getText(itemID)
  name = menu~itemTextToMethodName(text)

  say 'The constructed method name for this menu item:'
  say '  Item ID:   ' itemID
  say '  Item Text:' text
  say '  Method:    ' name
  say

return name
```

28.3.42. releaseMenuHandle

```
>>--releaseMenuHandle(--handle--)---------------------------------------------><
```

Releases (frees) the operating system resources used by a menu. The menu is specified by its Windows *handle*. To release the operating system resources for an ooDialog **Menu** object, use the *destroy*() method.

Arguments:

The single argument is:

handle [required]

The handle of the menu to release. The handle must not be null.

Return value:

Returns true for success, false for failure.

Remarks:

When the operating system frees the system resource used by a menu, that menu is no longer valid. If there is a Rexx menu object that represents the menu specified by the *handle* argument then the Rexx object's menu will no longer be valid. In general, the Rexx programmer will not have much, if any, use for this method. It is provided for completeness. Normally, the *destroy* method will be used.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

28.3.43. removeItem

```
>>--removeItem(--id--+---------------+--)--------><
                     +-,-byPosition--+
```

Removes (deletes) the specified menu command item from this menu.

Arguments:

The arguments are:

id [required]

The *ID* of the menu command item to remove. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

Returns true on success, false on error.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The specified menu item is not a menu command item.

Example:

This example is from an application that removes the Copy and Cut menu items from its menubar when there is currently no selected text. (A more common approach for this situation is to disable the two menu items.)

```
::method prepareMenu private
  expose menuBar

  ...
  if \ self~selectedText then do
    menuBar~removeItem(IDM_MI_COPY)
    menuBar~removeItem(IDM_MI_CUT)
  end

  ...
```

28.3.44. removePopup

```
>>--removePopup(--id--+---------------+--)-------><
                      +-,-byPosition--+
```

Removes, but does not delete, the specified popup menu from the menu.

Arguments:

The arguments are:

id [required]

The *ID* of the submenu to remove. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

Returns the specified submenu on success, 0 on failure.

Remarks:

When a popup menu (a submenu) is deleted, the operating system frees all resources used by the popup menu and any references to the popup are no longer valid. On the other hand, the *removePopup* method removes the specified submenu and returns it. The operating system resources are not freed and the returned submenu is still valid.

The returned submenu can then be used in any number of ways, as a context menu, it can be inserted into some other menu, or even reinserted back into the menu it was removed from. If the removed submenu is no longer needed by the application, then the *deletePopup* method can be used instead of *removePopup*.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The specified menu item is not a popup menu.

Example:

This example uses a context menu that is gotten from a menubar. Since the menubar is not used in the application, it is just the source for the context menu, the menubar is destroyed after the context menu is removed:

```
dlgPopup = menubar~removePopup(1, .true)

-- Start with the menu item to enable the list view greyed out.
dlgPopup~disable(IDM_RC_ENABLE_LV)

menuBar~destroy
```

28.3.45. removeSeparator

```
>>--removeSeparator(--id--+---------------+--)---><
                          +-,-byPosition--+
```

Removes (deletes) the specified menu separator from this menu.

Arguments:

The arguments are:

id [required]

The *ID* of the submenu to remove. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

Returns true on success, false on error.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The specified menu item is not a separator.

Example:

In this example, if there is no text selected the Cut and Copy menu items are removed from the menu. If there is no data on the clipboard then the Paste menu item is removed. Above and below those 3 menu items are separators. If all 3 menu items are removed then the separators are also removed.

```
::method prepareMenu private
  expose menuBar
  ...

  noSelection = .false
  noClipboard = .false

  if \ self~selectedText then do
    noSelection = .true
    menuBar~removeItem(IDM_MI_COPY)
    menuBar~removeItem(IDM_MI_CUT)
  end

  if \ self~hasClipboardData then do
    noClipboard = .true
    menuBar~removeItem(IDM_MI_PASTE)
  end

  if noSelection & noClipboard then do
    menuBar~removeSeparator(IDM_MI_SEPARATOR_TOP)
    menuBar~removeSeparator(IDM_MI_SEPARATOR_BOTTOM)
  end

  ...
```

28.3.46. setAutoConnection

```
>>--setAutoConnection(--onOff--+----------+--)---><
                               +-,-mName--+
```

Set the current status of *autoconnection*.

Arguments:

The arguments are:

onOff [required]

True to set autoconnection on, false to turn it off.

mName [optional]

This argument is ignored unless *onOff* is true. If so the *mName* argument can be used to change the current autoconnection method name.

When the autoconnection method name is not set, then when auto connection is done, the name of the connected method for each menu command item is composed from the menu item *itemTextToMethodName*. On the other hand, if the autoconnection method name is set, then each menu command item is connected to the same method, which is the autoconnection method name.

When the *setAutoConnection* method is used to turn autoconnection on, if the *mName* argument is omitted, then no change is made to the current setting for the autoconnection method name. If *mName* is the empty string, then the autoconnection method name, if there is one, is removed. If *mName* is not ommitted and not the empty string, then the autoconnection method name is set to *mName*.

Return value:

Returns true on success, false on error. An error is unlikely.

Remarks:

When *setAutoConnection* is used to turn autoconnection off, the autoconnection method name, if there is one, is always removed. If autoconnection is later turned on, and an autoconnection method name is desired, it will have to be specified.

Example:

This example turns autoconnection on and sets the autoconnection method name to onSelect. Thus, when autoconnection is done, every menu command item in the menu will be connected to the onSelect method:

```
menuBar~setAutoConnection(.true, onSelect)
```

This example turns autoconnection off:

```
menuBar~setAutoConnection(.false)
```

This example assumes autoconnection is already on, with the autoconnection method name set. However, the programmer now wants each menu command item connected to an individual method in the dialog. That is, the programmer wants the autoconnection method name removed:

```
menuBar~setAutoConnection(.true, '')
```

28.3.47. setHelpID

```
>>--setHelpID(--id--+------------+--)------------><
                    +-,-recurse--+
```

Sets the Help context identifier for the this menu, and optionally all submenus.

Arguments:

The arguments are:

id [required]

The help *ID* to be set. May be *symbolic* or numeric.

recurse [optional]

If true, set the id for this menu and all submenus of this menu. The default is false.

Return value:

Returns true on success and false on failure.

Remarks:

Associated with each menubar, menu, submenu, and shortcut menu is a help identifier. If the user presses the F1 key while the menu is active, this value is sent to the owner window as part of the Windows *connectHelp* event notification. The menu help ID is sent to the event handler as the first argument, the *id* argument.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example sets the help ID for 3 of the submenus in a menubar. Since none of the submenus have submenus, the *recurse* argument is omitted. The application uses symbolic IDs for all resource IDs:

```
...
menuBar~getPopup(IDM_POP_FILE)~setHelpID(IDM_HELP_FILE)
menuBar~getPopup(IDM_POP_EDIT)~setHelpID(IDM_HELP_EDIT)
menuBar~getPopup(IDM_POP_VIEW)~setHelpID(IDM_HELP_VIEW)
...
```

28.3.48. setID

```
>>--setID(--id-,-newID--+--------------+--)------><
                        +-,-byPosition-+
```

Sets or resets the resource ID of the specified menu item.

Arguments:

The arguments are:

id [required]

The *ID* of the menu item whose resource ID is to be set. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

newID [required]

The new resource ID for the specified menu item. May be *symbolic* or numeric.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.true`**.

Return value:

Returns true on success, false on error.

Remarks:

The *setID* method would usually be done using a by position ID to specify the menu item. Maybe to set the ID of a menu item that does not have an ID. However, the method could also be used to change the existing ID of an item. For this reason, the menu item can be specified by resource ID.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

28.3.49. setMaxHeight

```
>>--setMaxHeight(--height--+------------+--)----><
                           +-,-recurse--+
```

Sets the maximum height for this menu, and optionally all submenus.

Arguments:

The arguments are:

height [required]

The maximum height, in pixels, for this menu. Use zero to revert the height back to the system default, which is the height of the screen.

recurse [optional]

If true, set the height for this menu and all submenus of this menu. The default is false.

Return value:

Returns true on success, false on error.

Remarks:

When a menu reaches the maximum height, scroll bars are automatically added. The default height is the height of the screen. To revert back to the defualt height, specify 0 for the *height* argument.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example gets the current maximum height of the File drop down menu and checks if it is equal to the height of the dialog. If not, it changes the maximum height to the height of the dialog:

```
::method checkHeight private
  expose menuBar

  size = self~getRealSize

  filePopup = menuBar~getPopup(IDM_FILE)
  if filePopup~getMaxHeight <> size~height then filePopup~setMaxHeight(size~height)

  return 0
```

28.3.50. setText

```
>>--setText(--id--,--text--+---------------+--)--><
                           +-,-byPosition--+
```

Sets the text of the specified menu item. Separator menu items do not have text.

Arguments:

The arguments are:

id [required]

The *ID* of the menu item whose text is being set. May be *symbolic* or numeric. This can be a position ID or a resource ID, depending on the value of *byPosition*.

text [required]

The text for the menu item.

byPosition [optional]

If **`.true`**, *id* is a positional ID, otherwise it is a resource ID. The default is **`.false`**.

Return value:

Returns true on success and false on error.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The specified menu item is a separator.

28.3.51. uncheck

```
>>--uncheck(--IDs--+--------------+--)-----------><
                   +-,-byPosition-+
```

Unchecks one or more menu items. Separators can not be checked.

Arguments:

The arguments are:

IDs [required]

The item *ID(s)* to be unchecked, may be *symbolic* or numeric. This can be a single ID or a collection of IDs. The ID(s) can be position IDs or resource IDs, depending on the value of *byPosition*. However, if this is a collection of IDs there must be all the same type, all resource IDs or all position IDs.

byPosition [optional]

If true, the item IDs are positional IDs, otherwise they are resource IDs. The default is false, resource IDs.

Return value:

True on success, otherwise false.

Remarks:

To uncheck more than 1 menu item at a time, use a collection object, such as an **`array`**, where each item in the collection is the menu item ID of the item to be checked.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The menu item specified to be unchecked is a separator.

Example:

This example is from an editor application that has a hexadecimal view of the text. The user can switch back and forth between the hexadecimal and normal views using the menu item with the text of *hex*. When the user selects the menu item, the view is toggled. A check mark by the menu item shows it is in hex view, when there is no check mark the view is normal:

```
::method initDialog
  expose inHex

  inHex = .false
  ...

::method onHexSelect unguarded
  expose menuBar inHex

  if inHex then do
    inHex = .false
    menuBar~uncheck(IDM_HEX_VIEW)
    self~switchToAscii
  end
  else do
    inHex = .true
    menuBar~check(IDM_HEX_VIEW)
    self~switchToHex
  end

  return 0
```

28.3.52. unhilite

```
>>--unhilite(--IDs--+--------------+--)----------><
                    +-,-byPosition-+
```

Removes the high light from one or more menu items. Separators can not be high lighted. A high lighted menu item is drawn as though it were the selected item.

Arguments:

The arguments are:

IDs [required]

The menu item *ID(s)* whose high light is to be removed, may be *symbolic* or numeric. This can be a single ID or a collection of IDs. The ID(s) can be position IDs or resource IDs, depending on the value of *byPosition*. However, if this is a collection of IDs there must be all the same type, all resource IDs or all position IDs.

byPosition [optional]
: If true, the item IDs are positional IDs, otherwise they are resource IDs. The default is false, resource IDs.

Return value:

True on success, otherwise false.

Remarks:

To remove the high light from more than 1 menu item at a time, use a collection object, such as an **`array`**, where each item in the collection is the menu item ID of the item whose high light will be removed.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:
ERROR_INVALID_FUNCTION (1)
: Incorrect function. **Meaning:** The menu item specified to have its high light removed is a separator.

28.4. The Menu Bar Object

A menu bar is often called the top-level menu. It is the bar positioned just below the title bar of an application window or a dialog. The menu bar object is used, like the *Menu*, *dialog*, and dialog *control* objects as an abstract concept to document the ooDialog menu objects that represent menu bars. The menu bar classes in ooDialog are the *BinaryMenuBar*, *UserMenuBar*, and *ScriptMenuBar* classes.

The menu bar object is implemented through the **`MenuBar`** mixin class and all menu bars have the methods of the **`MenuBar`** class. Similar to the menu object, the menu bar object is essentially the **`MenuBar`** mixin class. Note that menu bars *are* menus. All menu bar objects have all the methods of the *Menu* object in addition to the methods specific to the menu bar.

28.4.1. Method Table

The following table lists the instance methods of the menu bar object.

Table 28.3. MenuBar Object Method Reference

Method	Description
attachTo	Attaches the menu bar to the specified dialog.
detach	Detaches this menu bar from its assigned dialog.
isAttached	Determines if this menu bar is currently attached to a dialog.
redraw	Tells the dialog this menu bar is attached to, to redraw the menu.
replace	If this menu bar is attached to a dialog, replaces this menu bar with the specified menu bar.

28.4.2. attachTo

```
>>--attachTo(--dlg--)-------------------------------><
```

Attaches this menu bar to the specified dialog.

Arguments:

The only argument is:

dlg [required]

The Rexx dialog the menu bar is to be attached to.

Return value:

True on success, false for failure.

Remarks:

If auto *connections* is turned on, (this can be checked through the *getAutoConnectStatus* method,) then all menu command items are connected to a method in the *dlg* dialog at this time.

A menu bar can only be attached to a dialog after the *underlying* dialog exists. A menu bar can only be attached to one dialog, and a dialog can only have one menu bar attached to it. If any of these conditions are not met, this method fails.

If this menu bar is already attached to a dialog and needs to be attached to a different dialog, first use the *detach* method to detach it. Then attach it to the other dialog. On the other hand, if the dialog already has a menu bar attached to it and this menu bar needs to be attached to that dialog get a reference to the dialog's current menu bar and use the *replace* method.

Details:

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** This menu is already attached to a dlg.

ERROR_NOT_ENOUGH_MEMORY (8)

Not enough storage is available to process this command. **Meaning:** Some menu items were not connected because the message table is full.

ERROR_INVALID_WINDOW_HANDLE (1400)

Invalid window handle. **Meaning:** The underlying Windows dialog does not exist for the specified *dlg* argument.

ERROR_INVALID_MENU_HANDLE (1401)

Invalid menu handle. **Meaning:** This menu has been destroyed.

ERROR_WINDOW_NOT_DIALOG (1420)

The window is not a valid dialog window. **Meaning:** The *dlg* argument is not a dialog object.

ERROR_INVALID_WINDOW_STYLE (2002)

The window style or class attribute is invalid for this operation. **Meaning:** The *dlg* argument is already attached to a menu bar.

When this method returns false, the menu is not attached to a dialog, except in one circumstance. If the **`.systemErrorCode`** is ERROR_NOT_ENOUGH_MEMORY, then the menu is attached to the dialog, but some menu items were not connected.

Example:

This example attaches a dialog's menu bar in the *initDialog* method. In the example, the menu bar was instantiated previously, and saved in an instance variable:

```
::method initDialog
```

```
expose menuBar

menuBar~attachTo(self)
...
```

28.4.3. detach

```
>>--detach---------------------------------------><
```

Detaches this menu bar from its assigned dialog.

Arguments:

This method has no arguments.

Return value:

Returns true on success, false on error.

Details:

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** This menu bar is not attached to a dialog.

Example:

This example detaches the menu bar from the dialog and destroys it as it is no longer needed in the application:

```
...
menuBar~detach
menuBar~destroy
...
```

28.4.4. isAttached

```
>>--isAttached-----------------------------------><
```

Determines if this menu bar is currently attached to a dialog.

Arguments:

This method takes no arguments.

Return value:

Returns true if this menu bar is attached to a dialog, otherwise false.

Details:

Sets the *.systemErrorCode*. There are no errors that could changes the value from 0.

28.4.5. redraw

```
>>--redraw---------------------------------------><
```

Tells the dialog this menu bar is attached to, to redraw the menu.

Arguments:

There are no arguments for this method.

Return value:

Returns true on success, false on error.

Details:

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** This menu bar is not attached to a dialog.

28.4.6. replace

```
>>--replace(--newMenuBar--)----------------------><
```

If this menu bar is attached to a dialog, the menu bar for the dialog is replaced by the specified menu bar.

Arguments:

The single required argument is:

newMenuBar [required]

The new menu bar for the dialog this menu bar is attached to.

Return value:

On success, returns the menu bar that was previously attached to the dialog, which of course is this menu bar. On error the **`.Nil`** object is returned.

Details:

Sets the *.systemErrorCode* on error. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** This menu bar is not attached to a dialog.

Example:

This example comes from a program that uses 2 completely different menu bars depending on if the application is in edit or view mode. When the mode is switched, the menu bar is replaced by the appropriate one for the mode:

```
::method onToggleMode
  expose fileMenu viewMenu currentMode

  if currentMode == 'Edit' then do
    fileMenu~replace(viewMenu)
    currentMode = 'View'
  end
  else do
    viewMenu~replace(fileMenu)
    currentMode = 'Edit'
  end
```

28.5. BinaryMenuBar Class

A **`BinaryMenuBar`** is a menu *MenuBar* whose menu template comes from a compiled binary resource, or whose menu template is constructed as an empty menu.

The ooDialog framework provides three basic types of menu bars. The only difference in the three types is the source of the menu template for the menu bar. In a *ScriptMenuBar* the source of the menu template is a resource *script* file (usually a .rc file) that ooDialog parses and uses to construct the menu template. For an *UserMenuBar*, ooDialog constructs the menu template dynamically from method invocations in the program code.

In general it is much easier to dynamically construct the menu template by starting with an empty **`BinaryMenuBar`** and inserting the desired menu items than it is to define the menu template in an **`UserMenuBar`**.

28.5.1. Method Table

The following table lists the class and instance methods of the **`BinaryMenuBar`**. For convenience, the inherited methods from the *MenuBar* mixin class are also listed here. Note that the **`BinaryMenuBar`** *is* a menu and therefore also contains all the methods of the *Menu* class.

Table 28.4. Method Reference

Method	Description
new	Instantiates a new binary menu bar object.
attachTo	Attaches the menu bar to the specified dialog.
detach	Detaches this menu bar from its assigned dialog.
isAttached	Determines if this menu bar is currently attached to a dialog.
redraw	Tells the dialog this menu bar is attached to, to redraw the menu.
replace	If this menu bar is attached to a dialog, replaces this menu bar with the specified menu bar.

28.5.2. new (Class)

```
>>--new(-+----------+-+------+-+-------+-+-----------+-+--------+-+---------+-)--><
         +--menuSrc-+ +-,-id-+ +-,-hID-+ +-,-attachTo-+ +-,-auto-+ +-,-mName-+
```

Instantiates a new **`BinaryMenuBar`** object.

The new menu bar is initially empty if *menuSrc* is omitted. If *menuSrc* is not omitted, the menu template is obtained from the compiled binary resource pointed to by *menuSrc*.

When an empty menu bar is instantiated, the programmer can add to the menu dynamically using menu object methods such as *insertPopup*, *insertItem*, etc.. Dynamically constructing a menu in this way is much less error prone than using the *UserMenuBar* class.

Arguments:

The arguments are:

menuSrc [optional]

The source of the menu template. If this argument is omitted an empty menu template is constructed in memory. When not omitted, the argument can be one of the following objects:

.nil:

Constructs an empty menu template in memory. This is identical to omitting the arugment altogether.

.ResDialog:

The menu template is located in the resource only DLL of the *ResDialog* object specified. In this case, the resource *id* argument is required.

.String:

The file name of an executable module containing the menu template. When this source of the menu template is used, the resource *id* argument is required. An executable module is either a DLL, or a .exe file, both of which can contain compiled resources.

.Pointer

A Rexx **`.Pointer`** object. If used, this argument must be a **`.Pointer`** object that represents a valid menu handle. Currently this source of the menu template is limited. The only way to get a menu handle is through the *getMenuHandle* method. However, future enhancements to ooDialog will likely have other ways to get a menu handle.

id [optional]

The resource ID of the menu bar, may be *symbolic* or numeric. This argument is required if *menuSrc* is either a **`.ResDialog`** or **`.String`** object.

hID [optional]

The context help resource ID of the menu bar, may be *symbolic* or numeric.

attachTo [optional]

If specified, attach this menu bar to the specified dialog. If used, *attachTo* has to be a *dialog* object and the dialog must be in a valid state to attach the menu bar. That is, the *underlying* dialog must exist and the dialog must not already have a menu bar attached. If any of these conditions are not met, an exception is raised.

auto [optional]

If true, turn on *auto connection*. The default is false. Refer to the discussion on command event *connection*s to understand the details of *auto connection*. When auto connection is on, *each* time this menu bar is attached to a dialog, all *menu command event*s are connected to a method in the dialog.

mName [optional]

This argument is ignored unless *auto* is true. If so *mName* is used as the auto connection method name for all menu command events. When *auto* is true and this argument is omitted, the name of the connected method for each menu command item is composed from the menu item *itemTextToMethodName*.

Return value:

Returns a new **`BinaryMenuBar`** object.

Remarks:

Raises conditions for all failures. If no condition is raised then the menu has been created successfully.

To use *symbolic* IDs with menu objects the programmer must use the global *.constdir*.

Details:

Raises syntax errors when incorrect arguments are detected, or other errors in instantiating the menu object have occurred.

Sets the *.systemErrorCode*. However, it will only be non-zero if the help ID is set and there is a Windows API failure. It is unlikely that this could happen.

Example:

This example loads a menu bar from a DLL, immediately attaches it to the dialog, and connects all menu command item events to the *onMenuSelect* even handler method:

```
  .application~useGlobalConstDir('O', 'resource.h')

  dlg = .BinaryDialog~new("menuResourceEx.dll", IDD_DLGMENU)
  ...

::method initDialog
  expose menuBar
  ...

  menuBar = .BinaryMenuBar~new("menuResourceEx.dll", IDR_MENU1, , self, .true,
 onMenuSelect)
```

28.6. ScriptMenuBar Class

The *ScriptMenuBar* is a menu *MenuBar* where the menu template is obtained from a resource *script* file.

The class has all the methods of the **`MenuBar`** and the *Menu* classes. The only difference between the **`ScriptMenuBar`** and the other menu bar classes such as the *BinaryMenuBar* or *UserMenuBar* is in the *new* method.

28.6.1. Method Table

The following table lists the class and instance methods of the **`ScriptMenuBar`**. For convenience, the inherited methods from the *MenuBar* mixin class are also listed here. Note that the **`ScriptMenuBar`** *is* a menu and therefore also contains all the methods of the *Menu* class.

Table 28.5. ScriptMenuBar Method Reference

Method	Description
new	Returns a new **`ScriptMenuBar`** object.
attachTo	Attaches the menu bar to the specified dialog.
detach	Detaches this menu bar from its assigned dialog.
isAttached	Determines if this menu bar is currently attached to a dialog.
redraw	Tells the dialog this menu bar is attached to, to redraw the menu.
replace	If this menu bar is attached to a dialog, replaces this menu bar with the specified menu bar.

28.6.2. new (Class)

```
>>--new(--rcFile--+------+-+----------+-+---------+-+-----------+-+------------+-)--><
```

```
                  +-,-id-+ +-,-helpID-+ +-,-count-+ +-,-connect-+ +-,-attachTo-+
```

Instantiates a new **`ScriptMenuBar`** object. The menu template is obtained from the resource script specified by the *rcFile* argument.

Arguments:

The arguments are:

rcFile [required]

The resource *script* file containing the menu definition for this script menu bar.

id [optional]

The resource ID of the menu definition in the resource script. May be *symbolic* or numeric. If this argument is omitted, the first menu definition found in the resource script is used.

helpID [optional]

The context help ID for this menu. May be numeric or *symbolic*. If omitted, the menu will not have a help ID.

count [optional]

The (approximate) count of menu items the menu will contain. The default when omitted is 200. This number is used to allocate the resources for the in-memory menu template. It is only approximate because the actual size of the template will vary with the length of the labels for the menu items.

When the template is being constructed, if the allocated resource is too small a syntax condition specifying the resource is too small will be raised. In this case, the programmer should use the *count* argument to specify a larger number of menu items.

connect [optional]

If true, each menu command item in the menu is connected to a method in a Rexx dialog object the *first* time the menu is attached to a dialog. This is a *connection* request type of connection. If the *attachTo* argument is used, then the menu command items are connected immediately. Otherwise the requests for connections are queued and the connections are made when the menu bar is actually attached to a dialog. The name of the connected method for each menu command item is composed from the menu item *itemTextToMethodName*. The default for this argument is false.

attachTo [optional]

Specifies a Rexx dialog object for the menu to attach to immediately. This argument must be a *dialog* object, the *underlying* Windows dialog must exist and that dialog must not already have a menu bar attached. If those conditions are not met, then a syntax condition is raised.

Return value:

The return is a new **`ScriptMenuBar`** object.

Remarks:

Raises conditions for all failures. If no condition is raised then the menu has been created successfully.

To use *symbolic* IDs with menu objects the programmer must use the global *.constdir*.

Details:

Raises syntax errors when incorrect arguments are detected, or other errors in instantiating the menu object have occurred.

Sets the *.systemErrorCode*. However, it will only be non-zero if the help ID is set and there is a Windows API failure. It is unlikely that this could happen.

Example:

This example creates a script menu bar and attaches it immediately to the dialog. Methods in the Rexx dialog are automatically connected to the menu command events, no help ID is assigned, and the menu item count is left at the default:

```
  .application~useGlobalConstDir("O", "ScriptMenuBar.h")

  dlg = .SimpleDialog~new("ScriptMenuBar.rc", IDD_SCRIPTMENUBAR_DLG)
  ...

::method initDialog
  expose menuBar

  menuBar = .ScriptMenuBar~new("ScriptMenuBar.rc", IDR_MENU_LV, , , .true, self)
  ...
```

28.7. UserMenuBar Class

A **`UserMenuBar`** is a *MenuBar* whose menu template comes from statements in the program code.

The ooDialog framework provides three basic types of menu bars. The only difference in the three types is the source of the menu template for the menu bar. In a *ScriptMenuBar* the source of the menu template is a resource *script* file (usually a .rc file) that ooDialog parses and uses to construct the menu template. For a *BinaryMenuBar* the menu template comes from a compiled binary resource, or the menu template is constructed as an empty menu.

In general it is much easier to dynamically construct the menu template by starting with an empty *BinaryMenuBar* and inserting the desired menu items than it is to define the menu template in an **`UserMenuBar`**.

28.7.1. Method Table

The following table lists the class and instance methods of the **`UserMenuBar`**. For convenience, the inherited methods from the *MenuBar* mixin class are also listed here. Note that the **`UserMenuBar`** *is* a menu and therefore also contains all the methods of the *Menu* class.

Table 28.6. UserMenuBar Method Reference

Method	Description
new	Returns a new **`UserMenuBar`**object.
addItem	Adds a command menu item to the menu template.
addPopup	Adds a popup menu, a submenu, to the menu template.
addSeparator	Adds a separator menu item to the menu template.
attachTo	Attaches the menu bar to the specified dialog.
complete	Finalizes the in-memory template and converts it to an actual menu bar.
detach	Detaches this menu bar from its assigned dialog.
isComplete	Tests if the menu template has been finished and converted to a menu.
isAttached	Determines if this menu bar is currently attached to a dialog.
redraw	Tells the dialog this menu bar is attached to, to redraw the menu.

Method	Description
replace	If this menu bar is attached to a dialog, replaces this menu bar with the specified menu bar.

28.7.2. new (Class)

```
>>--new(--+-----+-+---------+-+--------+-+-------+-+--------+--)----------><
          +--id-+ +-,-helpID-+ +-,-count-+ +-,-auto-+ +-,-mName-+
```

Instantiates a new **`UserMenuBar`** object. The menu template of a user menu bar is created dynamically in the program code, using the methods of the **`UserMenuBar`** class. This is similar to the creation of the dialog template in a *UserDialog*.

Arguments:

The arguments are:

id [optional]

The resource ID for the menu bar. May be *symbolic* or numeric. If this argument is omitted or is -1, the menu bar will not have a resource ID.

helpID [optional]

The context help ID for this menu. May be numeric or *symbolic*. If omitted, the menu will not have a help ID.

count [optional]

The (approximate) count of menu items the menu will contain. The default when omitted is 200. This number is used to allocate the resources for the in-memory menu template. It is only approximate because the actual size of the template will vary with the length of the labels for the menu items.

When the template is being constructed, if the allocated resource is too small a syntax condition specifying the resource is too small will be raised. In this case, the programmer should use the *count* argument to specify a larger number of menu items.

auto [optional]

If true, turn on *auto connection*. The default is false. Refer to the discussion on command event *connection*s to understand the details of *auto connection*. When auto connection is on, *each* time this menu bar is attached to a dialog, all *menu command event*s are connected to a method in the dialog.

mName [optional]

This argument is ignored unless *auto* is true. If so *mName* is used as the auto connection method name for all menu command events. When *auto* is true and this argument is omitted, the name of the connected method for each menu command item is composed from the menu item *itemTextToMethodName*.

Return value:

Returns a new **`UserMenuBar`** object.

Remarks:

Properly constructing a menu template using the **`UserMenuBar`** is somewhat difficult to do correctly, making it prone to error. It is much simpler to construct the menu template using an empty *BinaryMenuBar*. ooDialog programmers are encouraged to use the **`BinaryMenuBar`** class rather than the **`UserMenuBar`** class.

Raises conditions for all failures. If no condition is raised then the menu has been created successfully.

To use *symbolic* IDs with menu objects the programmer must use the global *.constdir*.

Details:

Raises syntax errors when incorrect arguments are detected, or other errors in instantiating the menu object have occurred.

Sets the *.systemErrorCode*. However, it will only be non-zero if the help ID is set and there is a Windows API failure. It is unlikely that this could happen.

Example:

This example shows the creation of a user menu bar in a private method of the dialog. The application use symbolic menu IDs. The default count of menu items is changed to 20 when the user menu bar is instantiated and auto connection is turned on. Since no method name is provided, each menu command event will be connected to a separate method.

```
  -- All resource IDs are symbolic and are defined in userMenu.h
  .application~useGlobalConstDir('O', "userMenu.h")
  ...

::method defineMenu private
  expose menuBar statusText

  count = 20
  menuBar = .UserMenuBar~new(IDH_MENU1, , count, .true)

  menuBar~addPopup(IDM_POP_FILES, "Files", "END", IDH_FILES)
    menuBar~addItem(ID_FILES_COPY, "Copy", "RADIO")
    menuBar~addItem(ID_FILES_MOVE, "Move", "DEFAULT CHECK RADIO")
    menuBar~addItem(ID_FILES_DELETE, "Delete", "RADIO")
    menuBar~addSeparator(IDM_SEP_FILES)
    menuBar~addPopup(IDM_POP_RENAME, "Rename", , IDH_RENAME)
      menuBar~addItem(ID_RENAME_LOWERCASE, "Lower case")
      menuBar~addItem(ID_RENAME_UPPERCASE, "Upper case", "CHECK END")
    menuBar~addSeparator(IDM_SEP_FILES)
    menuBar~addItem(ID_EXIT, "Exit", "END")

  if \ menuBar~complete then do
    statusText = 'User menu bar completion error: ' || -
                 .SystemErrorCode SysGetErrortext(.SystemErrorCode)
    return .false
  end

  return .true
```

28.7.3. addItem

```
>>--addItem(--id-,-text--+---------+--+---------+--)------------><
                         +-,-opts--+  +-,-mName-+
```

Adds a menu command item to the menu template. This method is only valid after the **`UserMenuBar`** has been instantiated and before the menu template is *complete*.

Arguments:

The arguments are:

id [required]
: The resource ID for the menu command item. May be *symbolic* or numeric.

text [required]
: The text, the label, for the menu item.

opts [optional]
: A list of 0 or more of the following keywords separated by spaces, case is not significant. The keywords set the state and type of the popup menu, and specify when the popup menu is the last item at the current level in the menu.

The keywords: ENABLED UNHILITE NOTDEFAULT UNCHECKED, although valid keywords, are simply ignored for this method. By default the state will be ENABLED UNHILITE NOTDEFAULT UNCHECKED, with no special type. The keywords are:

CHECKED	HILITE	RIGHTJUSTIFY
DEFAULT	MENUBARBREAK	END
DISABLED	MENUBREAK	
GRAYED	RADIO	

CHECKED
: A menu item state specification. Checks the menu item. If the item also has the RADIO type, the check is a bullet, otherwise it is a normal check mark.

DEFAULT
: A menu item state specification. Specifies that the menu item is the default. A menu can contain only one default menu item, which is displayed in bold.

DISABLE
: A menu item state specification. Disables the menu item and grays it so that it cannot be selected. DISABLED and GRAYED are exactly the same.

GRAYED
: A menu item state specification. Disables the menu item and grays it so that it cannot be selected. DISABLED and GRAYED are exactly the same.

HILITE
: A menu item state specification. Highlights the menu item. Although the menu item can be created with this state, it would not normally be done. It gives the user the appearance the menu is selected, but it is not actually selected.

MENUBARBREAK
: A menu item type specification. Places the menu item on a new line (for a menu bar) or in a new column (for a drop down menu, submenu, or shortcut menu). For a drop down menu, submenu, or shortcut menu, a vertical line separates the new column from the old.

MENUBREAK
: A menu item type specification. Places the menu item on a new line (if in the top level of a menu bar) or in a new column (for a menu item in a drop down menu, submenu, or shortcut menu). For a drop down menu, submenu, or shortcut menu, the columns are *not* separated by a vertical line.

RIGHTJUSTIFY
: A menu item type specification. Right-justifies the menu item and any subsequent popup menus. This value is valid only if the menu item is in the top level of a menu bar. (Although it is not common to put a menu item in the top level of a menu bar, it can be done.)

END

Indicates that this is the last menu item at the current level of the menu. The END keyword *must* be present if the item is the last menu item at the current level, and can *not* be present otherwise. Incorrect placement of the END keyword is the most common problem programmers have when using the **`UserMenuBar`** class.

mName [optional]

A method name to connect the menu command event to. The default is to not connect the menu command item. This is a *connection* request type of connection. If this argument is used it can not be the empty string.

Return value:

Returns true on success, false on error.

Remarks:

It is suggested that the ooDialog programmer create an empty *BinaryMenuBar* and add the menu items to the menu bar using the *insert* methods, such as *insertPopup*. This provides all the functionality of the **`UserMenuBar`**, but is easier to use.

Details:

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The **`UserMenuBar`** has already been *isComplete*.

ERROR_INVALID_PARAMETER (87)

The parameter is incorrect. **Meaning:** When the *mName* argument is used, it can not be the empty string.

Example:

There is a good example of using the *addItem* method in the *example*) for the *new* method of the **`UserMenuBar`** class.

28.7.4. addPopup

```
>>--addPopup(--id--,--name--+--------+--+----------+--)----><
                            +-,-opts-+  +-,-helpID-+
```

Adds a popup menu, a submenu, to the menu template. This method is only valid after the **`UserMenuBar`** has been instantiated and before the menu template is *complete*.

Arguments:

The arguments are:

id [required]

The resource ID for the popup menu. May be *symbolic* or numeric.

name [required]

The text label of the popup menu.

opts [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. The keywords set the state and type of the popup menu, and specify when the popup menu is the last item at the current level in the menu.

The keywords: ENABLED UNHILITE NOTDEFAULT, although valid keywords, are simply ignored for this method. By default the state will be ENABLED UNHILITE NOTDEFAULT, with no special type. The keywords are:

DEFAULT	HILITE	RIGHTJUSTIFY
DISABLED	MENUBARBREAK	RIGHTORDER
GRAYED	MENUBREAK	END

DEFAULT
: A popup menu state specification. Specifies that the popup menu is the default. A menu can contain only one default menu item, which is displayed in bold.

DISABLE
: A popup menu state specification. Disables the popup menu and grays it so that it cannot be selected. DISABLED and GRAYED are exactly the same

GRAYED
: A popup menu state specification. Disables the popup menu and grays it so that it cannot be selected. DISABLED and GRAYED are exactly the same

HILITE
: A popup menu state specification. Highlights the popup menu. Although the popup menu can be created with this state, it would not normally be done. It gives the user the appearance the popup menu is selected, but it is not actually selected.

MENUBARBREAK
: A popup menu type specification. Places the popup menu on a new line (for a menu bar) or in a new column (for a drop down menu, submenu, or shortcut menu). For a drop down menu, submenu, or shortcut menu, a vertical line separates the new column from the old.

MENUBREAK
: A popup menu type specification. Places the popup menu on a new line (for a menu bar) or in a new column (for a drop down menu, submenu, or shortcut menu). For a drop down menu, submenu, or shortcut menu, the columns are *not* separated by a vertical line.

RIGHTJUSTIFY
: A popup menu type specification. Right-justifies the popup menu and any subsequent popup menus. This value is valid only if the popup menu is in a menu bar.

RIGHTORDER
: A popup menu type specification. Specifies that menus cascade right-to-left (the default is left-to-right). This is used to support right-to-left languages, such as Arabic and Hebrew.

END
: Indicates that this is the last menu item at the current level of the menu. The END keyword *must* be present if the popup menu is the last menu item at the current level, and can *not* be present otherwise. Incorrect placement of the END keyword is the most common problem programmers have when using the **`UserMenuBar`** class.

helpID [optional]
: The context help ID for the popup menu. May be *symbolic* or numeric.

Return value:

Returns true on success, false on error.

Remarks:

It is suggested that the ooDialog programmer create an empty *BinaryMenuBar* and add the menu items to the menu bar using the *insert* methods, such as *insertPopup*. This provides all the functionality of the **`UserMenuBar`**, but is easier to use.

Details:

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The **`UserMenuBar`** has already been *isComplete*.

Example:

There is a good example of using the *addPopup* method in the *example*) for the *new* method of the **`UserMenuBar`** class.

28.7.5. addSeparator

```
>>--addSeparator(--id--+---------+--)-----------><
                       +-,-opts--+
```

Adds a separator, which is a horizontal line, to the menu template. This method is only valid after the **`UserMenuBar`** has been instantiated and before the menu template is *complete*.

Arguments:

The arguments are:

id [required]

The resource ID for the separator. May be *symbolic* or numeric. -1 is accepted to indicate no ID is desired, however, there are many advantages to assigning valid IDs to separators.

opts [optional]

A list of 0 or more of the following keywords separated by spaces, case is not significant. The keywords set the few valid types of the separator, and specify when the separator is the last item at the current level in the menu. The keywords are:

MENUBARBREAK	RIGHTJUSTIFY
MENUBREAK	END

MENUBARBREAK

A separator type specification. Places the separator, and all menu items following it, on a new line (for a menu bar) or in a new column (for a drop down menu, submenu, or shortcut menu). For a drop down menu, submenu, or shortcut menu, a vertical line separates the new column from the old.

MENUBREAK

A separator type specification. Places the separator, and all menu items following it, on a new line (for a menu bar) or in a new column (for a drop down menu, submenu, or shortcut menu). For a drop down menu, submenu, or shortcut menu, the columns are *not* separated by a vertical line.

RIGHTJUSTIFY

A separator type specification. Right-justifies the separator and any subsequent menu items. This value is valid only if the separator is in a menu bar.

END

Indicates that this is the last menu item at the current level of the menu. The END keyword *must* be present if the separator is the last menu item at the current level, and can *not* be present otherwise. Incorrect placement of the END keyword is the most common problem programmers have when using the **`UserMenuBar`** class.

Return value:

Returns true on success and false on error.

Remarks:

The Microsoft documentation indicates that a separator can not be used in a menu bar, only in a submenu. However, on Windows 7 at least, separators can be used in the menu bar. No vertical line is shown, but the separator will take up a little space. The separator *type* keywords work as expected. I.e., if the separator is given the RIGHTJUSTIFY type, it and all subsequent menu items are right justified in the menu bar.

It is suggested that the ooDialog programmer create an empty *BinaryMenuBar* and add the menu items to the menu bar using the *insert* methods, such as *insertPopup*. This provides all the functionality of the **`UserMenuBar`**, but is easier to use.

Details:

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The **`UserMenuBar`** has already been *isComplete*.

Example:

There is a good example of using the *addSeparator* method in the *example*) for the *new* method of the **`UserMenuBar`** class.

28.7.6. complete

```
>>--complete-------------------------------------><
```

The *complete* method finalizes the in-memory template and converts it to an actual menu bar.

Arguments:

There are no arguments for this method.

Return value:

Returns true on success, false on error.

Remarks:

The menu bar can not be attached to a dialog prior to invoking the *complete* method. Likewise, no menu items can be added to the template using the *addItem*, *addPopup*, or *addSeparator* methods after the *complete* method is invoked. However, once the menu bar is complete, the *insert* methods of the menu object, such as the *insertPopup* method can all be used to insert menu items.

Details:

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)
Incorrect function. **Meaning:** The menu template has already been finalized.

Example:
The *example*) for the *new* method of the **`UserMenuBar`** class uses the *complete* method.

28.7.7. isComplete

```
>>--isComplete-----------------------------------><
```

Tests if the menu template has been finished and converted to a menu.

Arguments:
This method has no arguments

Return value:
Returns true if the menu template has already been finished and converted to a menu, otherwise returns false.

Remarks:
Once the menu template has been finalized and converted to an actual menu, menu items can not longer be added to the template.

Example:
This example shows a method that adds a separator to the menu template when invoked. It first checks that the menu template is still available to add menu items

```
::method maybeAddSeparator private
  use strict arg id, isEnd

  if self~isComplete then return .false

  if isEnd the do
    self~addSeparator(id, 'END')
  end
  else do
    self~addSeparator(id)
  end
```

28.8. PopupMenu Class

A popup menu is a *menu* as opposed to a *menu bar*. Menus are arranged in a hierarchy with the menu bar being the top-level of the hierarchy. Popup menus go by a variety of names depending on where they are located in the hierarchy, and their use. But, they are all the same no matter the name used.

When a popup menu is in the menu bar, it is often called a drop-down menu. When a popup menu is within another popup menu it is usually called a submenu. Menus can also exist that are not attached to a menu bar, in which case they are often called shortcut menus, context menus, or popup menus. This documentation tries to be consistent and use the term popup menu in all cases. Method names in the Menu classes always use *popup*.

28.8.1. ContextMenu Event Handler

Note that the event handler for the CONTEXTMENU event connected through the *connectContextMenu* class method or the *connectContextMenu* instance method, described in this section, is the same as the event handler *described*/> for the CONTEXTMENU event connected through the *connectMenuEvent* method. The event and event handler are the same no matter how the event connection is made.

The event handler for the context menu event is invoked when the user right-mouse clicks on a window, types SHIFT-F10 on the keyboard, or types the VK_APPS key on the keyboard. The VK_APPS key is the *Applications* key on a Natural keyboard. Use the *connectContextMenu* class method, or the *connectContextMenu* instance method to connect the context menu event.

The interpreter does not wait for the return from the event handler, so the method does not need to return a value. However, good practice would be to always return a value from an event handler. 0 would be a good value to return.

```
::method onContextMenu unguarded
  use arg hwnd, x, y

  return 0
```

Arguments:

The event handling method receives 3 arguments:

hwnd

The window *handle* of the window the user clicked the mouse on, or the window that has the focus if the user generated the context menu event with the keyboard. Note that this will only be the dialog window handle of the user clicks on the dialog background. Quite often it is going to be one of the dialog controls.

x

The X coordinate of the mouse position, in screen *coordinates*) coordinates, at the time of the mouse click. Or, -1 if the user generated the context menu event by using the keyboard.

y

The Y coordinate of the mouse position, in screen *coordinates*) coordinates, at the time of the mouse click. Or, -1 if the user generated the context menu event by using the keyboard.

Return:

0 makes a good return value.

Example

The following example shows the event handler method that is only invoked for a list view in the application. Note that since the application uses the *show* method to put up the context menu, the handling of the menu command item selection is done in a menu command event *handler*.

```
  if \ lvPopup~connectContextMenu(onListViewContext, self~newListView(IDC_LV)~hwnd) then
 do
    say 'Error connecting context menu. SystemErrorCode:' || -
    .SystemErrorCode SysGetErrortext(.SystemErrorCode)
  end
  ...

::method onListViewContext
  expose lvPopup listView
  use arg hwnd, x, y

  if x == -1, y == -1 then do
    -- The keyboard was used, not the mouse.  Position the context menu as
    -- at the lower right of the list view.
    rect = listView~windowRect
    x = rect~right - .SM~cxVScroll + 15
    y = rect~bottom - 15
  end

  -- pos is the point on the screen, in pixels, to place the context menu.
  pos = .Point~new(x, y)

  -- Show the menu.
  ret = lvPopup~show(pos)
  if ret == -1 then do
    say 'lvPopup~show() failed SystemErrorCode:' || -
    .SystemErrorCode SysGetErrortext(.SystemErrorCode)
  end
```

28.8.2. Method Table

The following table lists the class and instance methods of the **PopupMenu**.

Table 28.7. PopupMenu Method Reference

Method	Description
Class Methods	
connectContextMenu	Connects the Windows context menu event with a method in the Rexx dialog.
new	Returns a new **PopupMenu**object.
Instance Methods	
assignTo	Assigns this popup menu to a dialog.
connectContextMenu	Connects the Windows context menu event with a method in the Rexx dialog.
isAssigned	Determines if this menu has an assigned owner.
show	Shows a popup menu as a context menu and returns immediately.
track	Shows a popup menu as a context menu and returns when the menu is dismissed

28.8.3. connectContextMenu (Class)

```
>>--connectContextMenu(--dlg--,--methodName--+---------+--+-----------+--)-----><
                                             +-,-hwnd--+  +-,-handles-+
```

Connects the context menu *event* notification to a method in a Rexx dialog.

Arguments:

The arguments are:

dlg [required]

The Rexx dialog the notification is being connect to.

methodName [required]

The method in the Rexx dialog to be invoked when the event happens.

hwnd [optional]

A window handle to filter the right-clicks on. This can be the window handle of any control in the dialog being connected, or even the dialog window handle itself. If used, only right-clicks on the specified window will be received. If omitted, all right-clicks on the dialog are received.

Return value:

Returns true on success, false on error.

Remarks:

The context menu event is generated when the user right-mouse clicks on a window. The event can also be generated by the keyboard by using SHIFT-F10 or the VK_APPS key (the Applications key on a Natural keyboard.) The *ContextMenu Event Handler* section has information on how to code the event handler.

In many cases, the *connectContextMenu* instance method will be more convenient to use. This class method is provided for times when the menu object itself is not available. For instance, some Windows applications dynamically create a context menu when receiving the event notification and then discard the menu.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on error. The system error code is set this way by ooDialog, in addition to error codes the OS might set:

ERROR_WINDOW_NOT_DIALOG (1420)

The window is not a valid dialog window. **Meaning:** The *dlg* argument is not a **`PlainBaseDialog`**, (or subclass of course.)

ERROR_NOT_ENOUGH_MEMORY (8)

Not enough storage is available to process this command. **Meaning:** The dialog message table is full.

If the programmer does not provide a method named *methodName* in the Rexx dialog, a syntax condition will be raised if any context menu events happen.

The underlying dialog receives the WM_CONTEXTMENU message as the notification for this event.

Example:

In this example the application creates the context menus after receiving the event notification. It needs to connect the event before the menu objects actually exist.

```
::method initDialog
  ...

  lvHwnd = self~newListView(IDC_LV_EMPLOYEES)
  .PopupMenu~connectContextMenu(self, "onListViewContext", lvHwnd)

  .PopupMenu~connectContextMenu(self, "onDialogContext", self~hwnd)

  ...
```

28.8.4. new (Class)

```
>>--new(--+----+--+----------+--)---------------><
```

```
         +-id-+  +-,-helpID--+
```

Instantiates a new **PopupMenu** object. Initially the menu is empty.

Arguments:

The arguments are:

id [optional]

The resource ID for the popup menu. May be *symbolic* or numeric. If this argument is omitted or is -1, the popup menu will not have a resource ID.

helpID [optional]

The context help ID for this menu. May be numeric or *symbolic*. If omitted, the popup menu will not have a help ID.

Return value:

Returns a new **PopupMenu** object.

Remarks:

When using a popup menu as a context menu, it is often convenient to simply get the menu from an existing menu through the *getPopup* method. This is particularly true when the application is based on resource script files or binary compiled resources. In those types of applications, the menus are created in the resource script file. Within the resource script, all menus have to be defined as menu bars. To define a popup menu that is only used as a context menu, simply define a menu bar that only contains one menu.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*. However, the variable will only be non-zero if the Windows API used to set the help ID fails. This is unlikely to happen.

Example:

This example instantiates a popup menu and then adds a few menu items to it:

```
m = .PopupMenu~new(IDM_LV_BAR)

m~insertItem(IDM_LV_RESTORE, IDM_LV_RESTORE, "Restore Original Order")
m~insertItem(IDM_LV_RESTORE, IDM_LV_PROFESSION, "Order by Profession")
m~insertItem(IDM_LV_PROFESSION, IDM_LV_FNAME, "Order by First Name")
```

28.8.5. assignTo

```
>>--assignTo(--dlg--+-------+-+----------+--)----><
                    +-,-auto+ +-,-mName--+
```

Assigns this popup menu to a dialog. This is only used when the menu is used as a context, or shortcut, menu.

Arguments:

The arguments are:

dlg [required]

The owner dialog to assign this menu to. See the Remarks section below.

auto [optional]

If true, turn on auto *connection*. The default is false. When *auto connection* is on, *each* time this popup menu is assigned to a dialog, all *menu command event*s are connected to a method in the dialog.

mName [optional]

Only used if *auto* true. If *auto* is true and this argument is not omitted, then *mName* is used to set, or reset, the auto connection method name. Note that this implies that if *auto connection* is already turned on for this menu, then *mName* can be used to change the auto connection method name.

Return value:

Returns true on success, false on error.

Remarks:

When a popup menu is used as a shortcut (context) menu it sends messages to its owner dialog when menu items are selected. Even if the short cut menu is only used with the *track* method, (where selecting a menu item does not send a message to its owner dialog,) the operating system still requires that the short cut menu have an owner dialog.

It is not neccessary to assign a short cut menu to a dialog, it just may be more convenient. If the shortcut menu does not have an assigned dialog, then the owner dialog must be specified each time the *track* or *show* methods are invoked.

A short cut menu can only be assigned to one dialog, if this menu is already assigned to a dialog, the owner dialog is replaced by the specified dialog.

Details:

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_MENU_HANDLE (1401)

Invalid menu handle. **Meaning:** This menu has been destroyed.

ERROR_WINDOW_NOT_DIALOG (1420)

The window is not a valid dialog window. **Meaning:** *dlg* is not a *dialog* object.

Example:

This example calls a function that creates a context menu and then assigns the menu to the dialog object:

```
contextMenu = createListViewMenu()

if \ contextMenu~assignTo(self, .true) then do
  say 'Error assigning the context menu. SystemErrorCode:' || -
  .SystemErrorCode SysGetErrortext(.SystemErrorCode)
end
```

28.8.6. connectContextMenu

```
>>--connectContextMenu(--methodName--+--------+--+---------+--)----------------><
                                     +-,-dlg--+  +-,-hwnd--+
```

Connects the context menu *event* notification to a method in a Rexx dialog.

Arguments:

The arguments are:

methodName [required]

The method in the Rexx dialog to be invoked when the event happens.

dlg [optional]

By default, the notification is connected to the owner dialog of the menu. However, the programmer can specify any Rexx dialog here to connect the notification to. If the menu does not have an owner dialog then the programmer must specify the Rexx dialog.

hwnd [optional]

A window handle to filter the right-clicks on. This can be the window handle of any control in the dialog being connected, or even the dialog window handle itself. If used, only right-clicks on the specified window will be received. If omitted, all right-clicks on the dialog are received.

Return value:

Returns true on success, false on error.

Remarks:

The context menu event is generated when the user right-mouse clicks on a window. The event can also be generated by the keyboard by using SHIFT-F10 or the VK_APPS key (the Applications key on a Natural keyboard.) The *ContextMenu Event Handler* section has information on how to code the event handler.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on error. The system error code is set this way in addition to what the OS might set:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The dialog argument was omitted and the menu does not have an assigned dialog.

ERROR_WINDOW_NOT_DIALOG (1420)

The window is not a valid dialog window. **Meaning:** The *dlg* argument is not a **`PlainBaseDialog`**, (or subclass of course.)

ERROR_NOT_ENOUGH_MEMORY (8)

Not enough storage is available to process this command. **Meaning:** The dialog message table is full.

If the programmer does not provide a method named *methodName* in the Rexx dialog, a syntax condition will be raised if any context menu events happen.

The underlying dialog receives the WM_CONTEXTMENU message as the notification for this event.

Example:

In this example the context menu notification is connected to the method, *onContextMenu*. All

```
::method initDialog
  ...

  lvHwnd = self~newListView(IDC_LV_EMPLOYEES)~hwnd
  .PopupMenu~connectContextMenu(self, "onListViewContext", lvHwnd)

  .PopupMenu~connectContextMenu(self, "onDialogContext", self~hwnd)
```

```
...
```

28.8.7. isAssigned

```
>>--isAssigned-----------------------------------><
```

Determines if this menu has an assigned owner.

Arguments:

This method has no arguments.

Return value:

Returns true if this menu has been assigned to an owner dialog, otherwise false.

Remarks:

Only popup menus can be assigned to a dialog, menu bars are *attached* to dialogs. Only popup menus that are used as context menus need to be *assignTo* to a dialog.

Details:

Resets the *.systemErrorCode* to 0. There are no operating system errors that would change that.

Example:

This example checks if the context menu is already assigned to a dialog before assigning it to the current dialog object:

```
if \ contextMenu~isAssigned then do
  if \ contextMenu~assignTo(self, .true) then do
    say 'Error assigning the context menu. SystemErrorCode:' || -
    .SystemErrorCode SysGetErrortext(.SystemErrorCode)
  end
end
```

28.8.8. show

```
>>--show(--p--+-------+--+--------+--+-------------+--+---------------+--)----><
              +-,-dlg-+  +-,-opts-+  +-,-bothButtons-+  +-,-excludeRect-+
```

Displays a context menu and returns after the user has selected an item or canceled the menu. When the user selects an item, a *menu command event* notification, using the id of the menu item selected, is generated. If the user cancels, nothing happens.

Arguments:

The arguments are:

p [required]

A *Point* object specifying the location for the shortcut menu, in screen *coordinates*) coordinates.

dlg [optional]

The Rexx dialog object that the menu should be assigned to. If the menu has already been assigned an owner dialog this argument can be omitted. If there is no already assigned dialog, this argument is required. Any menu displayed on the screen is required by the operating system to have an assigned dialog.

opts [optional]

A list of 0 or more of the following keywords separated by spaces, specifying additional options for the shortcut menu, case is not significant:

LEFT	BOTTOM	VERPOSANIMATION
HCENTER	HORNEGANIMATION	HORIZONTAL
RIGHT	HORPOSANIMATION	VERTICAL
TOP	NOANIMATION	RECURSE
VCENTER	VERNEGANIMATION	LAYOUTRTL

LEFT

Centers the shortcut menu horizontally relative to the x coordinate specified by the *p* argument.

HCENTER

Positions the shortcut menu so that its left side is aligned with the x coordinate specified by the *p* argument.

RIGHT

Positions the shortcut menu so that its right side is aligned with the x coordinate specified by the *p* argument. Specify only one of the LEFT, HCENTER, or RIGHT keywords. RIGHT is the default if none of the keywords is used.

TOP

Positions the shortcut menu so that its top side is aligned with the Y coordinate specified by the *p* argument.

VCENTER

Centers the shortcut menu vertically relative to the y coordinate specified by the *p* argument.

BOTTOM

Positions the shortcut menu so that its bottom side is aligned with the y coordinate specified by the *p* argument. Specify only one of the TOP, VCENTER, or BOTTOM keywords. BOTTOM is the default if none of the keywords is used.

HORNEGANIMATION

Animates the menu from right to left.

HORPOSANIMATION

Animates the menu from left to right.

NOANIMATION

Displays menu without animation.

VERNEGANIMATION

Animates the menu from bottom to top.

VERPOSANIMATION

Animates the menu from top to bottom.

HORIZONTAL

If the menu cannot be shown at the specified location without overlapping the excluded rectangle, the OS tries to accommodate the requested horizontal alignment before the requested vertical alignment.

VERTICAL

If the menu cannot be shown at the specified location without overlapping the excluded rectangle, the OS tries to accommodate the requested vertical alignment before the requested horizontal alignment.

RECURSE

Used to display a menu when another menu is already displayed. This is intended to support context menus within a menu.

LAYOUTRTL

Requires Common Control *Library* version 6.0 or later. Use right to left text layout. By default left to right text layout is used.

bothButtons [optional]

If true, the user can use either the right or the left buttons to select a menu item. If false only the left button can be used. The defualt if omitted is false.

excludeRect [optional]

A *Rect* object that specifies an area of the screen the menu should not overlap.

Return value:

Returns `.true` on success and `.false` on error.

Remarks:

The *show* and the *track* methods are both used to display a shortcut menu. The only difference is that when the *show* method is used, a menu command event is generated when the user selects a menu command item, but the *track* method directly returns the ID of the menu item selected.

For any animation to occur, the System Parameters Information menu animation must be on. (*SPI* class, *menuAnimation* attribute.) In addition, all the animation flags, except NOANIMATION, are ignored if menu fade animation is on. (*SPI* class, *menuFade* attribute.)

Microsoft says that it is essential that the System Metric, menu drop alignment (*SM* class, *menuDropAlignment* attribute,) be checked to determine the correct horizontal alignment flag (LEFT or RIGHT) and/or horizontal animation direction flag (HORPOSANIMATION or HORNEGANIMATION) to use. The say it is essential for creating the optimal user experience, especially when developing Microsoft Tablet PC applications.

Microsoft also notes that to display a context menu for a notification icon, the current window must be the foreground window before invoking the *show* or *track* methods. Otherwise, the menu will not disappear when the user clicks outside of the menu or the window that created the menu (if it is visible).

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:
ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The dialog argument was omitted and the menu does not have an assigned dialog.

ERROR_INVALID_WINDOW_HANDLE (1400)
: Invalid window handle. **Meaning:** The window handle for the dialog could not be obtained.

ERROR_INVALID_MENU_HANDLE (1401)
: Invalid menu handle. **Meaning:** This menu has been destroyed, or is no longer valid for some reason.

ERROR_WINDOW_NOT_DIALOG (1420)
: The window is not a valid dialog window. **Meaning:** The *dlg* argument was specified, but the object is not a *dialog* object.

Example:

This example shows a context menu:

```
pos = .Point~new(x, y)

ret = dlgPopup~show(pos, self)
if ret == -1 then do
  say 'dlgPopup~show() failed SystemErrorCode:' || -
  .SystemErrorCode SysGetErrortext(.SystemErrorCode)
end

return 0
```

28.8.9. track

```
>>--track(--p--+-------+--+--------+--+-------------+--+---------------+--)---><
               +-,-dlg-+  +-,-opts-+  +-,-bothButtons-+  +-,-excludeRect-+
```

Displays a context menu and returns the menu ID that the user has selected. When the user selects an item, *no* *menu command event* notification is generated. If the user cancels the menu 0 is returned.

Arguments:

The arguments are:

p [required]
: A *Point* object specifying the location for the shortcut menu, in screen *coordinates*) coordinates.

dlg [optional]
: The Rexx dialog object that the menu should be assigned to. If the menu has already been assigned an owner dialog this argument can be omitted. If there is no already assigned dialog, this argument is required. Any menu displayed on the screen is required by the operating system to have an assigned dialog.

opts [optional]
: A list of 0 or more of the following keywords separated by spaces, specifying additional options for the shortcut menu, case is not significant:

LEFT	BOTTOM	VERPOSANIMATION
HCENTER	HORNEGANIMATION	HORIZONTAL
RIGHT	HORPOSANIMATION	VERTICAL
TOP	NOANIMATION	RECURSE
VCENTER	VERNEGANIMATION	LAYOUTRTL

LEFT
: Centers the shortcut menu horizontally relative to the x coordinate specified by the *p* argument.

HCENTER
: Positions the shortcut menu so that its left side is aligned with the x coordinate specified by the *p* argument.

RIGHT
: Positions the shortcut menu so that its right side is aligned with the x coordinate specified by the *p* argument. Specify only one of the LEFT, HCENTER, or RIGHT keywords. RIGHT is the default if none of the keywords is used.

TOP
: Positions the shortcut menu so that its top side is aligned with the Y coordinate specified by the *p* argument.

VCENTER
: Centers the shortcut menu vertically relative to the y coordinate specified by the *p* argument.

BOTTOM
: Positions the shortcut menu so that its bottom side is aligned with the y coordinate specified by the *p* argument. Specify only one of the TOP, VCENTER, or BOTTOM keywords. BOTTOM is the default if none of the keywords is used.

HORNEGANIMATION
: Animates the menu from right to left.

HORPOSANIMATION
: Animates the menu from left to right.

NOANIMATION
: Displays menu without animation.

VERNEGANIMATION
: Animates the menu from bottom to top.

VERPOSANIMATION
: Animates the menu from top to bottom.

HORIZONTAL
: If the menu cannot be shown at the specified location without overlapping the excluded rectangle, the OS tries to accommodate the requested horizontal alignment before the requested vertical alignment.

VERTICAL
: If the menu cannot be shown at the specified location without overlapping the excluded rectangle, the OS tries to accommodate the requested vertical alignment before the requested horizontal alignment.

RECURSE
: Used to display a menu when another menu is already displayed. This is intended to support context menus within a menu.

LAYOUTRTL

Requires Common Control *Library* version 6.0 or later. Use right to left text layout. By default left to right text layout is used.

bothButtons [optional]

If true, the user can use either the right or the left buttons to select a menu item. If false only the left button can be used. The defualt if omitted is false.

excludeRect [optional]

A *Rect* object that specifies an area of the screen the menu should not overlap.

Return value:

Returns a non-negative number on success, -1 on error. On success, the return is either 0, meaning the user canceled the menu, or the id of the menu item selected.

Remarks:

The *track* and the *show* methods are both used to display a shortcut menu. The only difference is that when the *show* method is used, a menu command event is generated when the user selects a menu command item and when the *track* method directly returns the ID of the menu item selected.

For any animation to occur, the System Parameters Information menu animation must be on. (*SPI* class, *menuAnimation* attribute.) In addition, all the animation flags, except NOANIMATION, are ignored if menu fade animation is on. (*SPI* class, *menuFade* attribute.)

Microsoft says that it is essential that the System Metric, menu drop alignment (*SM* class, *menuDropAlignment* attribute,) be checked to determine the correct horizontal alignment flag (LEFT or RIGHT) and/or horizontal animation direction flag (HORPOSANIMATION or HORNEGANIMATION) to use. The say it is essential for creating the optimal user experience, especially when developing Microsoft Tablet PC applications.

Microsoft also notes that to display a context menu for a notification icon, the current window must be the foreground window before invoking the *show* or *track* methods. Otherwise, the menu will not disappear when the user clicks outside of the menu or the window that created the menu (if it is visible).

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The dialog argument was omitted and the menu does not have an assigned dialog.

ERROR_INVALID_WINDOW_HANDLE (1400)

Invalid window handle. **Meaning:** The window handle for the dialog could not be obtained.

ERROR_INVALID_MENU_HANDLE (1401)

Invalid menu handle. **Meaning:** This menu has been destroyed, or is no longer valid for some reason.

ERROR_WINDOW_NOT_DIALOG (1420)

The window is not a valid dialog window. **Meaning:** The *dlg* argument was specified, but the object is not a *dialog* object.

Example:

This example shows a context menu using the *track* method:

```
pos = .Point~new(x, y)

id = dlgPopup~track(pos, self)
if id == -1 then do
  say 'dlgPopup~track() failed SystemErrorCode:' || -
  .SystemErrorCode SysGetErrortext(.SystemErrorCode)
end
else do
  return self~doMenuCommand(ret
end
```

28.9. SystemMenu Class

The System menu is also known as the *Window* or *Control* menu. The System menu is normally defined and managed completely by the operating system. It is a pop-up menu that the user can open by clicking on the application icon in the title bar, or by right clicking anywhere on the title bar. The **`SystemMenu`** class provides methods for the programmer to work with and manipulate the Windows System menu.

The OS provides a standard set of menu items on the System menu that the user can choose to size or move a window, close a window, etc.. Menu items can be added, removed, and modified, but most applications just leave the System menu alone. A dialog window can have a System menu. For normal windows it is unusual to not have a System menu, but many simple dialogs do not use a System menu.

When a user selects a System menu item an *event* notification is sent to the owner window of the menu. However, most applications don't process the notification, but rather let the OS handle the notification with the default processing. The ooDialog programmer can connect the event notification to a method in her Rexx dialog in the same manner as other event notifications are connected. If the programmer adds menu command items to the System menu, the event notification for the item should be connected and processed by the application.

28.9.1. Method Table

The following table lists the constant, class, attribute, and instance methods of the SystemMenu.

Table 28.8. Method Reference

Method	Description
Constant Methods	The SystemMenu object provides a number of *constant* values through the **`::constant`**directive.
Class Methods	**Class Methods**
new (Class)	Instantiates and returns a new system menu object.
Instance Methods	**Instance Methods**
connectAllCommandEvents	.
connectCommandEvent	.
connectSomeCommandEvents	.
revert	.

28.9.2. Constant Methods

The **SystemMenu** object provides a number of *constant* values through the use of the **::constant** directive. The constants are listed and documented in this section. These constants are the *menu item IDs* for all System menu command items.

Recall that the constant methods defined by the **::constant** directive are both class and instance methods of the class they are defined in. The constants listed here are defined in the **SystemMenu** class. Therefore to access the constant value, the programmer uses either the **SystemMenu** class object, or an instantiated system menu object.

Note that for completeness, all the known system menu command items are defined and listed here. Microsoft does not explain what some of the command items are for. The constants can be used where ever a system menu command item identifier is needed. The constants provided by the system menu object are listed in the table below. The number in parentheses is the decimal value of the constant:

Table 28.9. SystemMenu Object Constant Reference

Constant Symbol	Description
SC_ARRANGE	Not explained. (61712)
SC_CLOSE	Closes the window. (61536)
SC_CONTEXTHELP	Changes the cursor to a question mark with a pointer. If the user then clicks a control in the dialog box, the control receives a WM_HELP message. (61824)
SC_DEFAULT	Selects the default item; the user double-clicked the window menu. (61792)
SC_HOTKEY	Activates the window associated with the application-specified hot key. (61776)
SC_HSCROLL	Scrolls horizontally. (61568)
SC_KEYMENU	Retrieves the window menu as a result of a keystroke. (61696)
SC_MAXIMIZE	Maximizes the window. (61488)
SC_MINIMIZE	Minimizes the window.. (61472)
SC_MONITORPOWER	Sets the state of the display. This command supports devices that have power-saving features, such as a battery-powered personal computer. (61808)
SC_MOVE	Moves the window. (61456)
SC_MOUSEMENU	Retrieves the window menu as a result of a mouse click. (61584)
SC_NEXTWINDOW	Moves to the next window. (61504)
SC_PREVWINDOW	Moves to the previous window. (61520)
SC_RESTORE	Restores the window to its normal position and size. (61728)
SC_SCREENSAVE	Executes the screen saver application specified in the [boot] section of the System.ini file. (61760)
SC_SEPARATOR	ID of the menu separator item. (61455)
SC_SIZE	Sizes the window. (61440)
SC_TASKLIST	Activates the Start menu. (61744)
SC_VSCROLL	Scrolls vertically. (61552)

28.9.3. new (Class)

```
>>--new(--dlg--)---------------------------------><
```

Instantiates a new **`SystemMenu`** object.

Arguments:

The single argument is:

dlg

The Rexx dialog object that contains the System menu. To be explicit, the dialog must have a System menu. A **`SystemMenu`** object can not be initialized if the dialog does not already have a System menu.

Return value:

Returns a new **`SystemMenu`** object.

Remarks:

The underlying menu object is a copy of the System menu of the specified dialog. Once the copy is made, the System menu can be manipulated and modified just like other menu objects.

Details:

Raises syntax errors when incorrect arguments are detected. Raises errors for all failures. If no errors are raised, the system menu object is valid.

Sets the *.systemErrorCode*.

Example:

This example instantiates a new **`SystemMenu`** object and then adds a couple of menu items to the system menu:

```
::method initDialog
  expose menu sysMenu

  -- Attach our menu bar to this dialog.
  if \ menu~attachTo(self) then self~error('Failed to attach menu')

  -- Get the system menu.
  sysMenu = .SystemMenu~new(self)

  -- Modify the system menu by inserting 3 menu items.  Each item is inserted
  -- before the fist item in the menu.  The .true arg specifies that the menu
  -- item ID (the first arg) is by position, rather than the default of a
  -- resource ID.
  sysMenu~insertSeparator(1, IDM_SYS_SEP1, .true)
  sysMenu~insertItem(1, IDM_SYS_REMOVE_EDIT, 'Remove Edit Menu', , , .true)
  sysMenu~insertItem(1, IDM_SYS_ADD_EDIT, 'Add Edit Menu', , , .true)
```

28.9.4. connectAllCommandEvents

```
>>--connectAllCommandEvents(--+-------+--+-------+--)-----------><
                              +--mth--+  +-,-dlg-+
```

Connects every command item in the system menu to a method, or methods, in the Rexx dialog.

Arguments:

The arguments are:

mth [optional]

Connect all menu command items to the method of this name. The default is to connect all menu command items to a method name composed from the text of the command item. If not omitted, *mth* can not be the empty string.

dlg [optional]

Connect the command items to the method(s) of this dialog object. The default is to connect the command items to the owner dialog of this menu. Since **`SystemMenu`** objects already have an owner dialog, there is no need to use this argument.

Return value:

Returns true on success, false on error.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The dialog argument was omitted and the menu does not have an assigned dialog.

ERROR_NOT_ENOUGH_MEMORY (8)

Not enough storage is available to process this command. **Meaning:** The dialog message table is full.

ERROR_WINDOW_NOT_DIALOG (1420)

The window is not a valid dialog window. **Meaning:** The dialog argument was not ommitted, but the *dlg* is not a dialog *dialog* object.

Example:

This example connects every menu command item in the system menu to the *onMenuSelect* method in the dialog:

```
::method connectSystemMenu unguarded private
  use strict arg sysMenu

  if \ sysMenu~connectAllCommandEvents(onMenuSelect) then self~error('Connect All Items
 Failed')
```

28.9.5. connectCommandEvent

```
>>--connectCommandEvent(--id--,--methodName--+--------+--)---------------------><
                                             +-,-dlg--+
```

Connects a system menu command item *event* notification with a method in a Rexx dialog. The notification is sent to the Windows dialog when a command item in a system menu is selected by the user.

Arguments:

The arguments are:

id [required]

The resource ID of the menu item, may be *symbolic* or numeric. The *constants* provided by the **`SystemMenu`** class can be used for the default items in a system menu.

methodName [required]

The name of the method in the Rexx dialog to connect the notification to. This can not be the empty string.

dlg [optional]

A Rexx dialog to connect the notification to. By default, the notification is is connected to the Rexx dialog that the menu is *attachTo* or *assignTo* to. If omitted and there is no assigned or attached dialog, no connection is made and the *.systemErrorCode* is set as described in the Details section. (To connect a menu item command event, there must be a dialog to connect it to.)

Return value:

Returns **`.true`** on success, otherwise **`.false`**. If there is an error, the **`.systemErrorCode`** is set as described in the Details section.

Remarks:

When using the *connectCommandEvent* method to connect menu command events, the programmer must use a menu item *ID*, not a menu item *position* identifier.

The *connectCommandEvent* method is used for all types of menus. However, the programmer should be aware of the differences in coding event *handler*s for **`SystemMenu`** events and other types of menus. See the *section*/> explaining the event handler for system menu command events.

Details:

Sets the *.systemErrorCode* on error. The system error code is set this way in addition to what the OS might set:

ERROR_INVALID_FUNCTION (1)

Incorrect function. **Meaning:** The dialog argument was omitted and the menu does not have an assigned dialog.

ERROR_INVALID_PARAMETER (87)

The parameter is incorrect. **Meaning:** The *methodName* argument can not be the empty string.

ERROR_WINDOW_NOT_DIALOG (1420)

The window is not a valid dialog window. **Meaning:** The *dlg* argument is not a **`PlainBaseDialog`**, (or subclass of course.)

ERROR_NOT_ENOUGH_MEMORY (8)

Not enough storage is available to process this command. **Meaning:** The dialog message table is full.

28.9.6. revert

```
>>--revert---------------------------------------><
```

Reverts the system menu for the dialog back to the standard system menu.

Arguments:

There are no arguments for this method.

Return value:

Returns **`.true`** on success, **`.false`** on error.

Remarks:

When a **`SystemMenu`** object is instantiated, it receives a copy of the system menu for the dialog. (The operating system maintains the standard system menu.) This method reverses that process. The operating system removes the copied system menu and replaces it with the standard system menu.

This object will then no longer be valid. *Note* however that any of the pre-defined System Command menu items that were connected to methods, will remain connected. Currently in ooDialog there is no way to 'unconnect' a method connection once it is made.

The operating system will only revert back to the standard system menu. A dialog that does not have minimize and maximize buttons is originally given a standard system menu, where the operating system has removed the menu items that are not applicable. Namely, the Restore, Size, Maximize and Minimize menu items.

When the operating system reverts back to the standard system menu on these dialogs, it does not remove these menu items. There is nothing the programmer can do about this, because only the operating system has access to the standard system menu.

Details:

Sets the *.systemErrorCode*.

Example:

This example reverts the system menu for the dialog back to the standard system menu.

```
::method onRevert
  expose sysMenu

  if \ sysMenu~revert then say 'revert() failed' .SystemErrorCode
 SysGetErrortext(.SystemErrorCode)
```

28.9.7. connectSomeCommandEvents

```
>>--connectSomeCommandEvents(--ids-+-------+-+-------------+-+-------+--)-----><
                                   +-,-mth-+ +-,-byPosition-+ +-,-dlg-+
```

Connects a collection of menu command items to a method, or methods, in the Rexx dialog.

Arguments:

The arguments are:

ids [required]

A **`Collection`** object containing the menu item *ID*s to connect. The IDs can be by position IDs or resource IDs, depending on the value of the *byPosition* argument. However, they must be all the same type, all resource IDs or all by position IDs.

mth [optional]
: Connect all menu command items to the method of this name. The default is to connect all menu command items to a method name composed from the text of the command item. If not omitted, *mth* can not be the empty string.

byPosition [optional]
: If true, the IDS are by positional IDs, otherwise the are resource IDs. The default is false, they are resource IDs.

dlg [optional]
: Connect the command items to the method(s) of this dialog object. The default is to connect the command items to the owner dialog of this menu. Since **`SystemMenu`** objects already have an owner dialog, there is no need to use this argument.

Return value:

Returns true on success, false on error.

Remarks:

The *ids* argument can be any ooRexx collection with a *makeArray* method that returns an array whose *items* are the menu item IDs to connnect.

This method quits when an error is detected. This implies that on an error return some menu items may have been connected.

Details:

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* on failure. In addition to error codes set by the operating system, the following error codes may be set by ooDialog:

ERROR_INVALID_FUNCTION (1)
: Incorrect function. **Meaning:** The dialog argument was omitted and the menu does not have an assigned dialog.

ERROR_NOT_ENOUGH_MEMORY (8)
: Not enough storage is available to process this command. **Meaning:** The dialog message table is full.

ERROR_INVALID_PARAMETER (87)
: The parameter is incorrect. **Meaning:** One or more of the specified item IDs is not a menu command item. Or the msg argument was used, but it is the empty string.

ERROR_WINDOW_NOT_DIALOG (1420)
: The window is not a valid dialog window. **Meaning:** The dialog argument was not ommitted, but the *dlg* is not a dialog *dialog* object.

Example:

This example connects the Close, Size, and Move command items in the System menu to 3 methods in the dialog. Note that those methods will be: closeAltF4(), size(), and move()

```
::method systemMenuConnectItems
  expose sysMenu
  use strict arg

  ids = .array~of(self~SC_CLOSE, self~SC_SIZE, self~SC_MOVE)

  if \ sysMenu~connectSomeCommandEvents(ids) then self~error('Connect Some Items Failed')
```

Resources

In the Windows OS, a *resource*, is binary data used by a Windows-based application. Usually, the binary data is attached to one of the application's executable files (*.exe or *.dll.) However, the binary data can also be generated dynamically in memory. (Which is common in ooDialog, for example the dialog template for a UserDialog is generated in memory.)

The data in standard resources describes things familar to ooDialog programmers like, dialog boxes, icons, menus, cursors, bitmaps, fonts, etc. The standard resources also include accelerator tables, string-table entries, message-table entries, and other resources that ooDialog does not currently have support for, but may support in the future.

This chapter describes ooDialog classes that provide access to Windows resources. The classes allow the oodialog programmer to use and manipulate resources in their ooDialog programs. This is an area of ooDialog that is slated for future improvements.

29.1. Resource Classes

The classes listed in the following table are documented in this chapter:

Table 29.1. ooDialog Resource Classes

Class	Description
Image *Image*	Objects used to work with and manipulate images.
ImageList *ImageList*	Objects used to efficiently manage large sets of images.
Mouse *Mouse*	Objects used to efficiently manage large sets of images.
ResourceImage *ResourceImage*	Objects used to represent modules that contain resources.

29.2. Image Class

The Image class is used to work with and manipulate images. Currently, the image types supported include bitmaps, icons, cursors (cursors are a type of icon), and enhanced metafiles. The enhanced metafile support is very limited at this time.

The class supports loading images from files, from resources contained in any executable files (*.exe and *.dll,) from the files associated with the ooDialog program, and from the generally available system resources.

Note: The `.Image` class is the future direction that ooDialog will take for working with images, including bitmaps. This is a more flexible approach than the older bitmap methods used when ooDialog was first developed. It will allow access to more of the modern features of the Windows user interface than the older approach does. The older bitmap methods were designed to work with Windows 3.1 and have a number of limitations. The ooDialog programmer is strongly encouraged to migrate her code towards the .Image class. The older bitmap methods should be considered deprecated, to a degree. Unfortunately, replacement methods for all of the older bitmap methods have not as yet been implemented. So, the older methods may still be necessary for some situations.

A loaded image takes up some small part of the systems's resources. It is common to release an image when the programmer is done with it. The .Image class has the *release* method to allow the programmer to release the image, if desired. It is **important** to note that when the ooDialog program ends, that is when the ooRexx interpreter process ends, the Windows operating system cleans up all the system resources associated with any images used in the ooDialog program. Not releasing images does no harm and the ooDialog programmer should not be unduly worried about this aspect of images.

In addition, when images are loaded as shared, the operating system completely manages them. Shared images should not be released. The .Image class tracks which images are loaded as shared and will not call the underlying API to release a shared image. So, again, the ooDialog programmer does not need to worry about mistakenly releasing a shared image. Once an image is released, it is no longer valid and the object can not be used as an argument to methods requiring a valid image. This applies to shared images also, even though they are not actually released.

Why then would the ooDialog programmer want to release images? The main reason would be to minimize the memory footprint of an application. In a normal ooDialog program, with five to ten images, releasing the images would have no noticeable impact on the memory footprint. However, in a long running Rexx program that opened and closed a lot of dialogs that used images, releasing the images as the dialogs were closed would make a difference in the long run. The operating system would not clean up the resources used by the images until the main Rexx program ended. If the main program ran for days, or maybe was never intended to be shut down, it would make sense to release images that were no longer needed.

29.2.1. Method Table

Instances of the `Image` class implement the methods listed in the following table:

Table 29.2. Methods of the `Image` Class

Method	Description
Class Methods	
colorRef	Provides a way to construct a COLORREF from the red, green, and blue values of a color.
fromFiles	Returns an array of `.Image` objects when given an array of file names.
fromIDs	Uses an array of resource IDs to instantiate and return an array of `.Image` objects.
getBValue	Returns the blue component of a RGB color.
getGValue	Returns the green component of a RGB color.
getImage	Instantiates a new `.Image` object from either an image file or from one of the system images.
getRValue	Returns the red component of a RGB color.
new	Image objects can not be instantiated from Rexx code using the *new*, other class methods are used.
toID	Used to translate a symbolic name to its integer value
userIcon	Returns a new image object from an icon resource added to the Rexx dialog object.
Instance Methods	
handle	Returns the Windows system handle to the image this object represents.
isNull	Tests if the image object is valid.
release	Releases the operating system resources used for the image.

Method	Description
systemErrorCode	Reflects any system error codes that are detected while working with an image object.

29.2.2. colorRef (Class Method)

```
>>-.Image~colorRef(--+---+--+-----+--+-----+--)-----------------><
                     +-R-+  +-,-G-+  +-,-B-+
```

The Windows API uses a COLORREF to specify a RGB color. This is a 32-bit number with a hexadecimal format of: **`0x00bbggrr`**. The **`colorRef`**() method provides a way to construct a COLORREF from the red, green, and blue values. Each color value (red, green, or blue) is a whole number in the range of 0 to 255 inclusive.

Windows also defines 2 special values for a COLORREF, CLR_NONE and CLR_DEFAULT. To get the numeric value for either of these values use CLR_NONE or CLR_DEFAULT for the R argument. E.g., **`ref = .Image~colorRef("CLR_NONE")`**.

Arguments:

All the arguments are optional, with 0 being the default for any omitted argument.

R

The red component of the RGB color, or one of the CLR_NONE / CLR_DEFAULT keywords. Case is not significant for either of the two keywords.

G

The green component of the RGB value.

B

The blue component of the RGB value.

Return value:

The return is a valid RGB color, specified as a COLORREF. Some method arguments related to images or colors require a COLORREF.

Example:

```
 ref = .Image~colorRef(245, 89, 255)
 say 'A fancy purple:' ref '(0x' || ref~d2x~right(8, '0')')'
 say

::requires 'oodPlain.cls'

/* Output:

A fancy purple: 16734709 (0x00FF59F5)

*/
```

29.2.3. fromFiles (Class method)

```
>>--fromFiles(--files--+--------+--+--------+--+---------+--)---><
                       +-,-type-+  +-,-size-+  +-,-flags-+
```

Gets an array of .Image objects, using an array of file names. This method is useful to load more than one image at a time, when all the images have the same specifications, type, size, and flags.

Arguments:

files

An array of file names to use to instantiate the .Image objects. The array can contain any number of file names, but it can not be sparse. That is, each index of the array must contain a file name. If an incorrect item is detected in the array, then a syntax error is raised and no images are returned.

On the other hand, if there is an error with the Win32 API loading an image, then no syntax error is raised. The index in the array for that image is left empty. One way to check for this type of error is to compare the number of items in the returned array with the number of items in file name array.

type

Specifies the type of the image, bitmap, icon, or cursor. You can use the *toID* method of the *Image* class to get the correct numeric value for one of the following symbols:

IMAGE_BITMAP	IMAGE_ICON
IMAGE_CURSOR	

The default is IMAGE_BITMAP.

size

A *Size* object that specifies the size of the image. The default is a size of 0x0. Under most circumstances this indicates that the actual size of the image should be used. However, the MSDN *documentation* documentation should be consulted for other meanings.

flags

The load resource flags for the LoadImage() API. The flags are one or more of the following symbols. You can use the *toID* method of the *Image* class to get the correct numeric value for any of the following symbols. The *or* method of the *DlgUtil* class can be used to combine more than one of the symbols if needed.

LR_DEFAULTCOLOR	LR_CREATEDIBSECTION
LR_DEFAULTSIZE	LR_LOADFROMFILE
LR_LOADMAP3DCOLORS	LR_LOADTRANSPARENT
LR_MONOCHROME	LR_SHARED
LR_VGACOLOR	

The default is LR_LOADFROMFILE.

Return value:

The method returns an array of .Image objects, one object for each image that was loaded from a file successfully.

Details:

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect arguments are detected.

Provides an interface to the Win32 API: **`LoadImage()`**. Use the MSDN *documentation* documentation to get more information on the arguments to this method.

Example:

```
```

29.2.4. fromIDs (Class method)

```
>>--fromIDs(--ids,--+--------+--+-------+--+---------+--)------><
                    +-,-type-+  +-,-size-+  +-,-flags-+
```

Uses an array of resource IDs to instantiate and return an array of .Image objects. This method loads the system image resources referenced by the resource IDs in the array. The images must all be the same type and size, and use the same load flags.

Arguments:

ids

An array of resource IDs to use to instantiate the .Image objects. The array can contain any number of IDs, but it can not be sparse. That is, each index of the array must contain a number. If an incorrect item is detected in the array, then a syntax error is raised and no images are returned.

On the other hand, if there is an error with the Win32 API loading an image, then no syntax error is raised. The index in the array for that image is left empty. One way to check for this type of error is to compare the number of items in the returned array with the number of items in ID array.

type

Specifies the type of the image: bitmap, icon, or cursor. You can use the *toID* method of the *Image* class to get the correct numeric value for one of the following symbols:

IMAGE_BITMAP	IMAGE_ICON
IMAGE_CURSOR	

The default is IMAGE_ICON

size

A *Size* object that specifies the size of the image. The default is a size of 0x0. Under most circumstances this indicates that the actual size of the image should be used. However, the MSDN *documentation* documentation should be consulted for other meanings.

flags

The load resource flags for the LoadImage() API. The flags are one or more of the following symbols. You can use the *toID* method of the *Image* class to get the correct numeric value for any of the following symbols. The *or* method of the *DlgUtil* class can be used to combine more than one of the symbols if needed.

LR_DEFAULTCOLOR	LR_CREATEDIBSECTION
LR_DEFAULTSIZE	LR_LOADFROMFILE
LR_LOADMAP3DCOLORS	LR_LOADTRANSPARENT
LR_MONOCHROME	LR_SHARED
LR_VGACOLOR	

The default is LR_SHARED | LR_DEFAULTSIZE.

Return value:

The method returns an array of .Image objects, one object for each system image that was loaded successfully.

Details:

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect arguments are detected.

Provides an interface to the Win32 API: **`LoadImage()`**. Use the MSDN *documentation* documentation to get more information on the arguments to this method.

Example:

```

```

29.2.5. getBValue (Class Method)

```
>>-.Image~getBValue(--colorRef--)----------------><
```

Returns the blue component of a RGB color.

Arguments:

The single argument is:

colorRef

A RGB color, a COLORREF. See the *colorRef* method for a brief discussion of these terms.

Return value:

The return is the blue component of the specified RGB color.

Example:

See the *example* for the *getRValue* method.

29.2.6. getGValue (Class Method)

```
>>-.Image~getGValue(--colorRef--)----------------><
```

Returns the green component of a RGB color.

Arguments:

The single argument is:

colorRef

A RGB color, a COLORREF. See the *colorRef*() method for a brief discussion of these terms.

Return value:

The return is the green component of the specified RGB color.

Example:

See the *example* for the *getRValue* method.

29.2.7. getImage (Class method)

```
>>--getImage(--id--+--------+--+--------+--+---------+--)-------><
                   +-,-type-+  +-,-size-+  +-,-flags-+
```

Instantiates a new .Image object from either an image file or from one of the system images. When id is a number then the corresponding system image is used. Otherwise, id is taken to be the name of an image file. File names can be either relative or absolute.

Arguments:

id

If not a number, then id must be the name of a stand-alone image file.

If id is a number than it is taken to be the resource id of an image provided by the system. The following are the symbolic names for all the system images. You can use the *toID* method of the *Image* class to get the correct numeric value for any of the following symbols:

IDI_APPLICATION	OCR_NORMAL	OBM_CLOSE	OBM_RGARROWD
IDI_HAND	OCR_IBEAM	OBM_UPARROW	OBM_LFARROWD
IDI_QUESTION	OCR_WAIT	OBM_DNARROW	OBM_DNARROW
IDI_EXCLAMATION	OCR_CROSS	OBM_RGARROW	OBM_COMBO
IDI_ASTERISK	OCR_UP	OBM_LFARROW	OBM_UPARROWI
IDI_WINLOGO	OCR_SIZENWSE	OBM_REDUCE	OBM_DNARROWI
	OCR_SIZENESW	OBM_ZOOM	OBM_RGARROWI
	OCR_SIZEWE	OBM_RESTORE	OBM_LFARROWI
	OCR_SIZENS	OBM_REDUCED	OBM_SIZE
	OCR_SIZEALL	OBM_ZOOMD	OBM_BTSIZE
	OCR_NO	OBM_RESTORED	OBM_CHECK
	OCR_HAND	OBM_UPARROWD	OBM_CHECKBOXES
	OCR_APPSTARTING	OBM_DNARROWD	OBM_BTNCORNERS

type

A number between 0 and 255 that specifies the type of the image: bitmap, icon, or cursor. You can use the *toID* method of the *Image* class to get the correct numeric value for one of the following symbols:

IMAGE_BITMAP	IMAGE_ICON
IMAGE_CURSOR	

The default is IMAGE_BITMAP.

size

A *Size* object that specifies the size of the image.

The default is a size of 0x0. Under most circumstances this indicates that the actual size of the image should be used. However, the MSDN *documentation* documentation should be consulted for other meanings.

flags

The load resource flags for the LoadImage() API. The flags are one or more of the following symbols. You can use the *toID* method of the *Image* class to get the correct numeric value for any of the following symbols. The *or* method of the *DlgUtil* class can be used to combine more than one of the symbols if needed.

LR_DEFAULTCOLOR	LR_CREATEDIBSECTION
LR_DEFAULTSIZE	LR_LOADFROMFILE
LR_LOADMAP3DCOLORS	LR_LOADTRANSPARENT
LR_MONOCHROME	LR_SHARED
LR_VGACOLOR	

When id specifies a file name, the default flags are LR_LOADFROMFILE, othewise the default flags are LR_SHARED | LR_DEFAULTSIZE. Note that the system images must be loaded as shared.

Return value:

A .Image object that represents the image specified. If an error happened, the object may not be valid. Use the *isNull* method to check if the image is valid. If there was an error, *.systemErrorCode* may help to determine the error.

Details:

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect arguments are detected.

Provides an interface to the Win32 API: **`LoadImage()`**. Use the MSDN *documentation* documentation to get more information on the arguments to this method.

Example:

```
flags = .DlgUtil~or(.Image~toID(LR_DEFAULTSIZE), .Image~toID(LR_SHARED), -
                    .Image~toID(LR_LOADMAP3DCOLORS))

questionIcon = .Image~getImage(.Image~toID(IDI_QUESTION),              -
                               .Image~toID(IMAGE_ICON),                -
                               .Size~new(0, 0), flags)
if questionIcon~isNull then do
  say 'Error getting the question icon.  Error code:' .systemErrorCode
end

/* Note that you can always use the raw numeric value for the args.
 * This works just as well:
 */
questionIcon = .Image~getImage(32514, 1, .Size~new(0, 0), 36928)
if questionIcon~isNull then do
  say 'Error getting the question icon.  Error code:' .systemErrorCode
end
```

29.2.8. getRValue (Class Method)

```
>>-.Image~getRValue(--colorRef--)----------------><
```

Returns the red component of a RGB color.

Arguments:

The single argument is:

colorRef

A RGB color, a COLORREF. See the *colorRef*() method for a brief discussion of these terms.

Return value:

The return is the red component of the specified RGB color.

Example:

```
  progressBar = self~newProgressBar("IDC_PB_FILES")
  if progressBar <> .nil then do
    purple = .Image~colorRef(245, 89, 255)

    say 'Going to set the background color for the progress bar'
    oldColor = progressBar~backgroundColor(purple)
```

```
        say 'Old background color was:' oldColor '(0x'oldColor~d2x')'
        say '  Red:   ' .Image~getRValue(oldColor)
        say '  Green:' .Image~getGValue(oldColor)
        say '  Blue: ' .IMage~getBValue(oldColor)
        say
      end

      /* Output might be:

      Going to set the background color for the progress bar
      Old background color was: 11460781 (0xAEE0AD)
        Red:   173
        Green: 224
        Blue:  174

      */
```

29.2.9. new (Class method)

The Image class does not allow new Image objects to be instantiated from Rexx code using the new() method. New Image objects are obtained through one of the other Image class methods, or they are returned from methods of other classes.

These methods are used to create new Image object(s).
Image (class) method:*getImage*
Image (class) method:*fromFiles*
Image (class) method:*fromIDs*
Image (class) method:*userIcon*
ResourceImage (instance) method:*getImage*
ResourceImage (instance) method:*getImages*

29.2.10. toID (Class method)

```
>>--toID(--symbolicName--)-----------------------><
```

The *toID* method is used to translate a symbolic name to its integer value. In general the symbolic name is related to images or color. Many of the arguments to the methods of classes related to images use the integer value of a symbolic ID in the Windows API. This method allows the programmer to use the symbolic ID without knowing what the actual integer value is.

Take for example the task of retrieving the 'Question' icon resource from the system using the *getImage* method. The ooDialog programmer could either use the numerical value of 32514 or use the *toID* method as follows. Note that the two invocations of getImage() are equivalent:

```

    qIcon = .Image~getImage(.Image~toID(IDI_QUESTION))

    qIcon = .Image~getImage(32514)
```

The symbolic ID keywords are spelled exactly as Microsoft spells them in the MSDN *documentation* which allows the ooDialog to easily look up the meaning of any single ID while at the same time reducing the documentation task for the ooDialog developers. The keywords are case sensitive and must be all in upper-case.

Arguments:

The single required argument is:

symbolicName

The symbolic name whose numeric value is desired. There any number of symbolic names and they are not listed here. Rather the symbolic names are listed in the documentation for the methods they are applicable to.

Return value:

The return value is the numeric value of the symbol.

Example:

```
say 'The numeric value of the IDI_WINLOGO symbol is:' .Image~toID(IDI_WINLOGO)
/*
  Output on the console would be:

  The numeric value of the IDI_WINLOGO symbol is: 32517
*/
```

29.2.11. userIcon (Class method)

```
>>--userIcon(--dlg--,--id--+---------+--+---------+--)----------><
                           +-,-size--+  +-,-opts--+
```

Returns a new **`Image`** object from an icon resource added to the dialog through the *addIconResource*() method.

Arguments:

The arguments are:

dlg [required]

The Rexx dialog object to which the user icon was added.

id [required]

The resource ID of the icon. Can be numeric or *symbolic*.

size [optional]

A *Size* object that specifies the size of the image.

If this argument is omitted, a size of 0 is used. Under most circumstances this indicates that the actual size of the icon should be used. However, if LR_DEFAULTSIZE is used as one of the flags in the *opts* argument, then the width and height specified by the system metric values for icons is used.

opts [optional]

A whole number that specifies the flags that control how the operating system loads the icon. The flags are one or more of the following symbols. You can use the *toID* method of the *Image* class to get the correct numeric value for any of the following symbols. The *or* method of the *DlgUtil* class can be used to combine more than one of the symbols if needed.

LR_DEFAULTCOLOR	LR_CREATEDIBSECTION
LR_DEFAULTSIZE	LR_LOADFROMFILE
LR_LOADMAP3DCOLORS	LR_LOADTRANSPARENT
LR_MONOCHROME	LR_SHARED
LR_VGACOLOR	

The default value is LR_LOADFROMFILE. The flags for this method, must include LR_LOADFROMFILE and can not contain LR_SHARED. If this argument is specified by the programmer, the value is checked, and corrected if necessary. See the remarks section for some additional comments.

Return value:

An image object, which may be a *isNull* image on error.

Remarks:

For most usage, the Rexx programmer would do one of the following things:

- Omit the *size* and *flags* arguments. This would load the icon using the actual size of the icon. If multiple icon images are included in the file, then the first image found would be used.
- Omit the *size* and include the LR_DEFAULTSIZE flag in the *flags*. This would load the using the system metric values for icons.
- Specify the size for the loaded image and omit the *flags* argument.

This method uses the LoadImage() Windows API to actually load the icon image. For more advanced uses of this method, consult the MSDN *documentation* documentation.

Details

Raises syntax errors when incorrect arguments are detected. This includes an invalid symbolic ID.

Sets the *.systemErrorCode*.

Example:

This example adds an icon resource to the dialog through the *addIconResource* method and then uses that icon for the image in a *Static* image control.

```
  dlg = .SimpleDialog~new("Simple.rc", IDD_DIALOG1, , 'Simple.h')
  if dlg~initCode = 0 then do
    dlg~addIconResource(IDI_ENHANCED_QUESTION_MARK, 'fancyQuestion.ico')
    dlg~execute("SHOWTOP", IDI_CORPORATE_IMAGE)
  end

::method initDialog
  expose iconControl

  iconControl = self~newStatic(IDC_ICON_QUESTION)
  if iconControl <> .nil then do
    questionMark = .Image~userIcon(self, IDI_ENHANCED_QUESTION_MARK, .Size~new(64, 64) )
    if \ questionMark~isNull then iconControl~setIcon(questionMark)
  end

  self~connectButtonEvent(IDC_PB_YES, "CLICKED", onYes)
  self~connectButtonEvent(IDC_PB_NO, "CLICKED", onNo)
```

29.2.12. handle

```
>>--handle---------------------------------------><
```

Returns the Windows system handle to the image this object represents. It is an error to invoke this method if the image is null, or after the image has been released.

Currently, the handle is only useful for display. In the ooDialog framework, methods that use images for arguments, use the .Image object, not the image handle. Older methods that use a bitmap handle for a argument will not work with this handle.

Arguments:

The method does not have an argument.

Return value:

The return is the handle to the image.

Example:

```
hIcon = .Image~getImage(.Image~toID(IDI_QUESTION), .Image~toID(IMAGE_ICON))
say 'hIcon:  ' hIcon
say ' handle:' hIcon~handle

/* Output to the console might be (on a 64-bit Windows system)

   hIcon:  an Image
   handle: 0x000000000001002B
*/
```

29.2.13. release

```
>>--release--------------------------------------><
```

Releases the image. This will free up operating system resources used for the image. See the introduction to the *Image* class for some discussion on releasing an image.

Once an image object has been released, it is an error to use the object. However, the *isNull* and *systemErrorCode* methods can always be used. The isNull() method will return **`.true`** after an image object has been released. Shared images should not be released, the operating system manages them. To prevent the programmer from accidentally releasing a shared image, the .Image object tracks which images are loaded as shared and does not release a shared image if the programmer requests it.

Arguments:

There are no arguments.

Return value:

The possible return values are:

0
: No error detected.

non-zero
: The operating system error code.

Details:

Sets the *.systemErrorCode*.

Example:

In this example, when the user presses the Test button, the current image for the static control is replaced by the system question mark image. The old image is no longer needed and it is released.

```
    ::method onTest

      iconControl = self~newStatic(IDC_ICON_QUESTION)
      if iconControl <> .nil then do
        hQuestion = .Image~getImage(.Image~toID(IDI_QUESTION), .Image~toID(IMAGE_ICON))
        say 'Question icon:' hQuestion~handle
        if hQuestion~isNull then do
          say 'errror code:' .systemErrorCode
        end
        else do
          image = iconControl~setImage(hQuestion)
          say 'Swapped images:'
          say '   new icon:' hQuestion~handle
          say '   old icon:' image~handle

          -- The old image is no longer needed.
          ret = image~release
          say 'Released old image return:' ret '.SystemErrorCode:' .systemErrorCode
        end
      end

      /* Output on the screen, on a 64-bit system, might be:

       Question icon: 0x000000000001002B
       Swapped images:
         new icon: 0x000000000001002B
         old icon: 0x0000000001120639
       Released old image return: 0 .SystemErrorCode: 0

      */
```

29.2.14. isNull

```
>>--isNull---------------------------------------><
```

The isNull() method tests if the image object is valid. The image will be null if an error occurred when it was loaded, or if the image has been released.

Arguments:

There are no arguments.

Return value:

The method returns **`.true`** if the image is not valid, is null, and **`.false`** if the image is not null.

Example:

In this example the programmer wanted to load the system question mark icon, but used the wrong image type. (He uses IMAGE_BITMAP instead of IMAGE_ICON.) The system will refuse to load the image. To test if the image was loaded okay, use the isNull() method. Note that the getImage() method set the .SystemErrorCode. As in all software, the error codes are not always that informative.

```
    hQuestion = .Image~getImage(.Image~toID(IDI_QUESTION), .Image~toID(IMAGE_BITMAP))
    if hQuestion~isNull then do
      say 'System Errror code:' .systemErrorCode
      say '  System message:  ' SysGetErrorText(.systemErrorCode)
    end
```

```
       /* Output would be:

       System Errror code: 1814
         System message:   The specified resource name cannot be found in the image file.

       */
```

29.2.15. systemErrorCode

```
>>--systemErrorCode------------------------------><
```

The systemErrorCode attribute of the image object will reflect any system error codes that are detected while working with an image object. This is the same error code as the *.systemErrorCode* is set to. This can be useful if the programmer wants to check for an error code at some point when it is possible that .systemErrorCode has been reset. (See the example below.)

Like the *isNull* method this method can be invoked even when the image is not valid. However, it should not be used to check if the image is valid because it is possible for the value to be 0 and the image to be null.

Arguments:

There are no arguments to this method

Return value:

The value is usually 0, but may be a system error code if one was detected.

Example:

In this example the system error code attribute for an image is checked at a point in the life-cycle of the application when checking the **`.systemErrorCode`** would be meaningless:

```
       ::method onYes
         expose hIcon

         -- Need to release the icon image, but it is not always valid.
         if \ hIcon~isNull then hIcon~release
         else do
           say 'The icon image is not valid.'
           say '  Error when it was loaded:' hIcon~systemErrorCode
           say '  System message:          ' SysGetErrorText(hIcon~systemErrorCode)
         end
         return self~ok:super

       /* Output might be:

       The icon image is not valid.
         Error when it was loaded: 2
         System message:           The system cannot find the file specified.

       */

       /* Note that the below was the error that caused the above.  The file name
        * is actually 'shaveIce.ico' not shavedIce.ico.
        */

         if iconControl <> .nil then do
           hIcon = .Image~getImage("shavedIce.ico", .Image~toID(IMAGE_ICON))
           ...
```

29.3. ImageList Class

An image list is a object used to efficiently manage large sets of images. Either icons or bitmaps. In an image list, all images are the same size and are accessed by a zero-based index. The images are in screen device format. Optionally, each image in the list can have a matching monochrome bitmap that is used as a mask to draw the image transparently.

The .ImageList object acts as an interface to the underlying operating system Image List, which Microsoft calls a control. Although Microsoft calls the Image List a control and documents it with the other dialog controls, it is not a control in the same sense as button, edit, or list-view controls. It is not a window, all dialog controls are windows. It does not send or receive window messages. Use the MSDN *documentation* documentation to get more information on exactly how Image Lists work.

When an image list is no longer needed, it can be released. This frees up the system resources used for the images and the image list. The programmer releases an image list by using the *release* method. Once an image list is released it can no longer be used. It is an error to invoke any method on the released object, except for the *isNull* method. The isNull() method can be invoked any time to test if an image list is valid or not. It should go without saying that a programmer should not release an image list that is in use.

Future versions of the .ImageList are intended to provide a complete interface to the underlying Image List control. At this time not all functionality is implemented.

29.3.1. Method Table

Instances of the ImageList class implement the methods listed in the following table:

Table 29.3. Image Instance Methods

Method	Description
Class Methods	
create	Creates a new empty image list of the type specified.
new	Currently it is not intended for the ooDialog programmer to instantiate an image list object directly using the new method.
Instance Methods	
add	Adds one or more bitmap images to the image list.
addIcon	Adds an icon or cursor image to the image list.
addImages	Adds a number of images to the image list.
addMasked	Adds one or more bitmap images, along with a mask to the image list.
duplicate	Creates a duplicate image list.
getCount	Determines the number of images in the image list.
getIcon	Creates and returns an icon image from an image and mask in this image list.
getImageSize	Determines the size of the images in the image list.
handle	Returns the Windows system handle to the image list this object represents.
isNull	Used to check if an image list is valid.
remove	Removes the image at the specified index from the image list.
removeAll	Removes all images from the image list.

Method	Description
release	Releases the resource module so that the operating system can reclaim the system resources used by the image list.

29.3.2. create (Class method)

```
>>--create(--+--------+-+---------+-+---------+-+--------+--)---><
             +--size--+ +-,-flags-+ +-,-count-+ +-,-grow-+
```

Creates a new empty image list of the type specified.

Arguments:

The arguments are:

size [optional]

A *Size* object that specifies the size of a single image in the image list. All images in an image list have the same size. The size is specified in pixels.

If this argument is omitted the system default size for an icon is used. This size can vary depending on which version of Windows is running and whether the user has selected to use large icons or not. A typical size is 32 x 32 (in pixels.)

flags [optional]

The flags that specify the type of image list to create. The flags are one or more of the following symbols, but can include only one ILC_COLOR* value. You can use the *toID* method of the *Image* class to get the correct numeric value for any of the following symbols. The *or* method of the *DlgUtil* class can be used to combine the symbols.

ILC_MASK	ILC_COLOR24
ILC_COLOR	ILC_COLOR32
ILC_COLORDDB	ILC_PALETTE
ILC_COLOR4	ILC_MIRROR
ILC_COLOR8	ILC_PERITEMMIRROR
ILC_COLOR16	

ILC_MASK

The image list contains two bitmaps, one of which is used as a mask. The mask is a monochrome bitmap. If this value is omitted, the image list contains only one bitmap.

ILC_COLOR

Indicates to use the default image list type when none of the other ILC_COLOR* flags are used. Usually the default is ILC_COLOR4. However, with older display drivers the default may be ILC_COLORDDB.

ILC_COLORDDB

The image list uses a device-dependent bitmap.

ILC_COLOR4

The image list uses a 4-bit, 16-color, device-independent bitmap, DIB, section for the bitmap

ILC_COLOR8

The image list uses an 8-bit DIB section. The color table use the same colors as the halftone palette.

ILC_COLOR16
: Use a 16-bit (32/64k-color) DIB section.

ILC_COLOR24
: Use a 24-bit DIB section.

ILC_COLOR32
: Use a 32-bit DIB section.

ILC_PALETTE
: Microsoft has this flag but it is not implemented. ooDialog accepts the flag in case it is implemented in future versions of Windows.

ILC_MIRROR
: Languages such as Hebrew or Arabic that read right-to-left can be mirrored by Windows. When an image list is created on a mirrored version of Windows, the images are mirrored. I.e.,, they are flipped to display from right to left. This flag on mirrored versions of Windows instructs to OS to not automatically mirror the images.

ILC_PERITEMMIRROR
: This flag is ignored unless ILC_MIRROR is also used. It should be used when the image list contains a strip of images.

count [optional]
: The initial size for the image list. The size is the number of images the image list contains. When this argument is omitted the default is 6.

grow [optional]
: The amount by which the image list can grow if the operating system needs to resize the image list to make room for more images. The default value when omitted is 0.

Return value:

A new, empty, image list is returned.

Note: It is theoretical possible for this method to fail and the returned image list to be null. However, in practice, it is virtually impossible to cause a failure. Using a size of 0 x 0 seems about the only way.

Details:

Raises syntax errors when incorrect arguments are detected.

Provides an interface to the **`ImageList_Create()`** API. Use the MSDN *documentation* documentation to get more information on the arguments to this method.

The ILC_MIRROR and ILC_PERITEMMIRROR flags require Common Control *Library* version 6.0 or later. If necessary use the *comCtl32Version*() method to determine the current version of the library.

Example:

```
-- We set the flags to create a 24 bit color, masked image list.
flags = .DlgUtil~or(.Image~toID(ILC_COLOR24), .Image~toID(ILC_MASK))

-- Create an empty .ImageList object:
imageList = .ImageList~create(.Size~new(61, 46), flags, 10, 10);
```

29.3.3. new (Class method)

Currently it is not intended for the ooDialog programmer to instantiate an image list object directly using the new method. This may change in the future and will be documented at that time, if the intention changes. .ImageList objects are instantiated using the *create* method.

29.3.4. add

```
>>--add(--image-+---------+--)-------------------><
                +-,-mask--+
```

Adds a bitmap image or images to the image list. The number of images is inferred from the width of the added bitmap. Optionally adds the mask for the bitmap(s). If the image list does not use a mask, the mask argument is ignored, even if it is present.

Internally the image list makes a copy of the bitmap. After the method returns, the original image can be released if it is not needed anymore.

Arguments:

The arguments are:

image

The .Image object that represents the image to add. This must be a bitmap image. The width of the image determines the number of images added. (Remember all images in a image list are the same size.)

mask

The mask(s) to use with the image(s). This must be a bitmap image.

Return value:

On success, the index of the first image added is returned, otherwise -1 is returned.

Details:

Raises syntax errors when incorrect arguments are detected or if the image list is null.

Provides an interface to the **`ImageList_Add()`** API. The MSDN *documentation* documentation can provide more information on this method.

Example:

This example sets the image list for a list-view control. The images are loaded from psdemolv.bmp which is a bitmap 16 pixels high by 64 pixels wide, containing 4 individual images. Since the image list is created with a size of 16 x 16, when the add() method is used, the image list infers that the bitmap is 4 images.

```
-- Set the images for the items in the list-view.
image = .Image~getImage("bmp\psdemolv.bmp")
imageList = .ImageList~create(.Size~new(16, 16), .Image~toID(ILC_COLOR8), 4, 0)
if \image~isNull,  \imageList~isNull then do
   imageList~add(image)
   lc~setImageList(imageList, .Image~toID(LVSIL_SMALL))

   -- The image list makes a copy of the bitmap, so we can release it now.
   image~release
end
```

29.3.5. addIcon

```
>>--addIcon(--image--)---------------------------><
```

Adds an icon or cursor image to the image list. If the image list is masked, then both the image and mask bitmaps of the icon or cursor are copied. If it is not masked, then only the image bitmap is copied.

Internally the image list makes a copy of the bitmap. After the method returns, the original image can be released if it is not needed anymore.

Arguments:

The only argument is:

image

The .Image object that represents the image to add. This must be an icon or cursor image.

Return value:

This method returns the index of the added icon or cursor on success, otherwise -1 is returned.

Details:

Raises syntax errors when incorrect arguments are detected or if the image list is null.

Provides an interface to the **`ImageList_AddIcon()`** API. The MSDN *documentation* documentation can provide more information on this method.

Example:

Here the system icon images are loaded and then displayed in a list-view. Each icon will show in the list-view with the text for the icon being its numeric resource ID. Since the system icons are shared, the icon images are not released after they are added to the image list.

```
ids = .array~new()
ids[ 1] = .Image~toID(IDI_APPLICATION)
ids[ 2] = .Image~toID(IDI_HAND)
ids[ 3] = .Image~toID(IDI_QUESTION)
ids[ 4] = .Image~toID(IDI_EXCLAMATION)
ids[ 5] = .Image~toID(IDI_ASTERISK)
ids[ 6] = .Image~toID(IDI_WINLOGO)

flags = .DlgUtil~or(.Image~toID(ILC_COLOR8), .Image~toID(ILC_MASK))

imageList = .ImageList~create(.Size~new(32, 32), flags, 20, 10)

do i = 1 to ids~items
  image = .Image~getImage(ids[i], .Image~toID(IMAGE_ICON))
  imageList~addIcon(image)
end

list~setImageList(imageList, .Image~toID(LVSIL_NORMAL))

do i = 1 to ids~items
  list~add(ids[i], i - 1)
end
```

29.3.6. addImages

```
>>--addImages(--images--+---------+--)-----------><
```

```
                        +-,-cRef--+
```

Adds a number of images to the image list. The images are supplied in a non-sparse array and must all be of the same type.

The images are added to the image list in the same order as they exist in the array. If an error occurs in the middle of processing the images, the method returns at that point. This means that images prior to the error will exist in the image list, but no images in the array after the error will be placed in the image list. Which images were placed in the image list can be determined by the return value of this method.

Details:

Raises syntax errors when incorrect arguments are detected or if the image list is null.

Arguments:

The arguments are:

images

An array of non-null .Image objects. The array must not be sparse, that is each index in the array must contain an .Image object. The images can be bitmaps, icons, or cursors, but each image in the array must be the same type. (Remember that cursors are icons, so that the array can contain a mixture of icons and cursors.) Bitmaps can not be mixed with icons or cursors.

cRef

A *colorRef* that is used to generate the mask if the image list is a masked image list. See the *addMasked* method for more details on this argument. This argument is ignored if the images are not bitmaps, or if the image list is not masked.

Return value:

This method returns the image list index of the last successfully added image, or -1 if no images were added.

Example:

This example creates an image list from a number icons and then displays them in a list-view control in a dialog.

```
list  = self~newListView(IDC_LV_IMAGES)

files1 = .array~new()
files1[ 1] = "Bee.ico"
files1[ 2] = "Camera.ico"
files1[ 3] = "Camera1.ico"
files1[ 4] = "Default.ico"
files1[ 5] = "Disabled.ico"
files1[ 6] = "Hot.ico"
files1[ 7] = "Lamp.ico"
files1[ 8] = "Mountain.ico"
files1[ 9] = "Normal.ico"
files1[10] = "Penguin.ico"
files1[11] = "Picture.ico"
files1[12] = "Pushed.ico"
files1[13] = "Question32.ico"
files1[14] = "Search32.ico"
files1[15] = "Skull.ico"
files1[16] = "Stolen.ico"
files1[17] = "Window.ico"

size = .Size~new(32, 32)
```

```
    images = .Image~fromFiles(files1, .Image~toID(IMAGE_ICON), size)
    if images~items <> files1~items then do
      say 'Error loading images.'
      say '  System error:' .systemErrorCode
      say '  Message:     ' SysGetErrorText(.systemErrorCode)
      return
    end

    count = images~items
    flags = .DlgUtil~or(.Image~toID(ILC_COLOR24), .Image~toID(ILC_MASK))

    imageList = .ImageList~create(size, flags, 20, 10);

    lastAdded = imageList~addImages(images)
    if lastAdded <> (count - 1) then do
      -- Not all images were added.  We just ignore this and display
      -- in the list-view what was added.
    end

    -- Set the image list for the list-view's normal icons.
    list~setImageList(imageList, .Image~toID(LVSIL_NORMAL))

    -- Add an item to the list-view for each image.  The
    -- text for each item will be the icon file name and
    -- the icon will be the image we loaded.
    do i = 0 to lastAdded
      list~add(files1[i + 1], i)
    end
```

29.3.7. addMasked

```
>>--addMasked(--image-,-cRef--)------------------><
```

Adds a bitmap image, or images, to the image list. The COLORREF cRef is used to generate the mask.

Internally the image list makes a copy of the bitmap. After the method returns, the original image can be released if it is not needed anymore.

Arguments:

The arguments are:

image

The .Image object that represents the image to add. This must be a bitmap image. The width of the image determines the number of images added. (Remember all images in a image list are the same size.)

cRef

The *colorRef* to use to generate the mask. In the added image, each pixel that matches this color is changed to black, and the corresponding bit in the mask is set to 1.

Return value:

On success, the index of the first added image is returned, otherwise -1 is returned.

Details:

Raises syntax errors when incorrect arguments are detected or if the image list is null.

Provides an interface to the **`ImageList_AddMasked()`** API. The MSDN *documentation* documentation can provide more information on this method.

Example:

This example comes from a dialog with a Tab control. An image list is used to set an icon for each tab. Each icon is a colored letter on a white background. Using a white color to generate the mask causes the image to be drawn transparently. The colored letter itself shows, and the rest of the image lets the underlying color show through.

```
    -- Add all the tabs, including the index into the image list for an icon for
    -- each tab.
    tc~addFullSeq("Red", 0, ,"Green", 1, , "Moss", 2, , "Blue", 3, ,   -
                  "Purple", 4, ,  "Cyan", 5, , "Gray", 6)

    -- Create a COLORREF (pure white) and load our bitmap.
    cRef = .Image~colorRef(255, 255, 255)
    image = .Image~getImage("bmp\psdemoTab.bmp")

    -- Create our image list, as a masked image list.
    imageList = .ImageList~create(.Size~new(16, 16),                    -
                                  .DlgUtil~or(.Image~toID(ILC_COLOR24),  -
                                  .Image~toID(ILC_MASK)),                -
                                   10, 0)

    if \image~isNull,  \imageList~isNull then do
       imageList~addMasked(image, cRef)
       tc~setImageList(imageList)

       image~release
    end
```

29.3.8. duplicate

```
>>--duplicate------------------------------------><
```

Creates a duplicate image list. All information in the original image list is copied to the new image list. (Overlay images are not copied, but ooDialog does not have support for image list overlay images at this time.) The two image lists are independent. Adding, or removing, images from one image list has no effect on the other.

Arguments:

There are no arguments.

Return value:

The return is a copy of the image list.

Details:

Raises a syntax error if the image list is null.

Provides an interface to the **`ImageList_Duplicate()`** API. The MSDN *documentation* documentation can provide more information on this method.

Example:

This fictious example is from a point-of-sale application for a restaurant. The customer is presented with a list-view of menu items, each list-view item has a colorful icon depicting the selection. The customers place their orders through the application and wait-people then bring them their meal when it is ready. The dinner menu has all the items that the lunch menu does, plus some more. The customer can choose whether to order from the lunch menu or the dinner menu, so the lunch menu image list has to remain unchanged.

```
::method getDinnerImageList private
  use strict arg lunchImageList

  dinnerList = lunchImageList~duplicate
  fileArray = self~getDinnerImageFiles
  dinnerList~addImages(fileArray)

return dinnerList
```

29.3.9. getCount

```
>>--getCount-------------------------------------><
```

Determines the number of images in the image list.

Arguments:

There are no arguments.

Return value:

Returns the number of images currently in the image list.

Details:

Raises a syntax error if the image list is null.

Provides an interface to the **`ImageList_ImageCount()`** API. The MSDN *documentation* documentation can provide more information on this method.

Example:

```
::method displayImageListCount private
  use strict arg imageList
  say 'Image list:' imageList~handle 'has' imagelist~getCount 'images.'

  /*
    Possible output on a Windows 64-bit system:

    Image list: 0x00000000000ED420 has 17 images.
  */
```

29.3.10. getIcon

```
>>--getIcon(--index--+-------------+--+----------+--)------------------------><
                     +-,-drawStyle--+  +-,-ovrLay--+
```

Creates and returns an icon image from an image and mask in this image list.

Arguments:

The arguments are:

index [required]

The one-based index of the image in this list.

drawStyle [optional]

A list of 0 or more of the following keywords separated by spaces. These keywords specify the drawing style that the returned icon is created with. Case is not significant. The default if this argument is omitted is NORMAL. Note that several of the keywords have the exact same meaning. This is intentional on Microsoft's part:

BLEND	FOCUS	SELECTED
BLEND25	MASK	TRANSPARENT
BLEND50	NORMAL	

BLEND

Draws the image, blending 50 percent with the system highlight color. This value has no effect if the image list does not contain a mask.

BLEND25

Draws the image, blending 25 percent with the system highlight color. This value has no effect if the image list does not contain a mask.

BLEND50

Draws the image, blending 50 percent with the system highlight color. This value has no effect if the image list does not contain a mask.

FOCUS

Draws the image, blending 25 percent with the system highlight color. This value has no effect if the image list does not contain a mask.

MASK

Draws the mask.

NORMAL

Draws the image using the background color for the image list. If the background color is the CLR_NONE value, the image is drawn transparently using the mask.

SELECTED

Draws the image, blending 50 percent with the system highlight color. This value has no effect if the image list does not contain a mask.

TRANSPARENT

Draws the image transparently using the mask, regardless of the background color. This value has no effect if the image list does not contain a mask.

ovrLay [optional]

The one-based index of the overlay image in this list. See the remarks section.

Return value:

Returns a valid icon *Image*, or the `.nil` object on error.

Remarks:

The overlay image is drawn transparently over the primary image, that is the icon, returned from this method. The overlay image must be added to the image list separately. ooDialog has not yet implemented a method for adding overlay images to the image list. That is intended for a future enhancement. Therefore, at this time, the *ovrlay* argument has no meaning.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example comes from an application that uses a resource only DLL. The DLL contains the dialog template for the application and a bitmap of images. The bitmap contains a series of 20 images side by side. Each image is 16 by 16 pixels. The bitmap is used to create an image list, then an icon can be retrieved from the image list for any of the images in the bitmap file using the *getIcon* method and the index of the image in the bitmap:

```
ri = .ResourceImage~new(self)
imageBMP = ri~getImage(IDI_ICONS_IL)

imageList = .ImageList~create(.Size~new(16, 16), 'COLOR32', 20, 0)
imageList~add(imageBMP)
imageBMP~release

icon = imageList~getIcon(1)
statusBar~setIcon(icon, 1)
```

29.3.11. getImageSize

```
>>--getImageSize---------------------------------><
```

Determines the size of the images in the image list. All images in any single image list have the same size.

Arguments:

There are no arguments.

Return value:

The size of the images in the image list is returned in a *Size* object.

Details:

Raises a syntax error if the image list is null.

Provides an interface to the **`ImageList_IconSize()`** API. The MSDN *documentation* documentation can provide more information on this method.

Example:

```

  ::method displayImageListSize private
    use strict arg imageList
    s = imageList~getImageSize
    h = s~height 'pixels high'
    w = s~width 'pixels wide'
    say 'Image list:' imageList~handle 'contains images' h 'by' w

    /*
      Possible output on a Windows 64-bit system:

      Image list: 0x00000000000DCB20 contains images 32 pixels high by 32 pixels wide
    */
```

29.3.12. handle

```
>>--handle---------------------------------------><
```

Returns the Windows system handle to the image list this object represents. It is an error to invoke this method if the image list is null, or after the image has been released.

At this time, the handle is only useful for display. In the ooDialog framework, methods that use image lists for arguments, use the .ImageList object, not the image list handle.

Arguments:

There is no argument to this method.

Return value:

The return is the image list handle.

Example:

```
::method displayImageList private
  use strict arg imageList
  say 'Currently using this image list:' imageList~handle

  /*
    Possible output on a Windows 64-bit system:

    Currently using this image list: 0x00000000000DCB20
  */
```

29.3.13. isNull

```
>>--isNull---------------------------------------><
```

Used to check if an image list is valid. This method can be invoked on any image list, even after it has been released. An image list will always be null after it has been released. It is conceivable that, if an error occurs during an image list creation, the returned image list might be null. However, this is extremely unlikely. Therefore, for all practical purposes, an image list will only be null after it has been released.

Arguments:

There are no arguments to this method.

Return value:

This method returns true if the image list is null, otherwise false.

Example:

This example comes from some test code.

```
-- See if we can create an image list with no size.
imageList = .ImageList~create(.Size~new(0, 0), .Image~toID(ILC_COLOR24),  20, 10)
if imageList~isNull then do
  say 'Can not create a 0 x 0 image list.'
```

```
      return .false
    end

    return .true

    /*
      Output will be:

      Can not create a 0 x 0 image list.

    */
```

29.3.14. release

```
>>--release--------------------------------------><
```

Releases the image list allowing the operating system to reclaim the resources used by the image list. Once the image list has been released, it can no longer be used.

Arguments:

There are no arguments to this method.

Return value:

This method always returns 0.

Example:

This example comes from a program that runs continuously. During the life-cycle of the application it displays and then closes a dialog that creates some image lists. The image lists are always different depending on the state of things. Each time the dialog closes, it releases the image lists it created to free up the operating system resources.

```
    ::method ok
      self~cleanUp
      return self~ok:super

    ::method cancel
      self~cleanUp
      return self~cancel:super

    ::method cleanUp private
      expose coolTempImages warningImages buildingStatusImages

      if coolTempImages \== .nil then coolTempImages~release
      if warningImages \== .nil then warningImages~release
      if buildingStatusImages \== .nil then buildingStatusImages~release
```

29.3.15. remove

```
>>--remove(--index--)----------------------------><
```

Removes the image at *index* from the image list. When the image is removed, all the indexes in the list are adjusted. I.e., if the image at index 2 is removed, the image at index 3 becomes index 2, the image at index 4 becomes index 3, etc..

Arguments:

The single argument is:

index

The one-based index of the image to remove.

Return value:

This method returns true on success, otherwise false.

Details:

Raises syntax errors when incorrect arguments are detected or if the image list is null.

Provides an interface to the **`ImageList_Remove()`** API. The MSDN *documentation* documentation can provide more information on this method.

Example:

This example comes from the fictious point-of-sale application for a restaurant. Sometimes the chef runs out of a menu item and the item is then removed from the list-view so that the customers can no longer order it. (Rigorous error checking is not done because the application was written by one of the waiters.)

```
::method removeFromMenu private
  use strict arg index

  menu = self~newListView(IDC_LV_MENU)

  bigIcons = menu~getImageList(.Image~toID(LVSIL_NORMAL))
  bigIcons~remove(index)

  smallIcons = menu~getImageList(.Image~toID(LVSIL_SMALL))
  smallIcons~remove(index)

  stateIcons = menu~getImageList(.Image~toID(LVSIL_STATE))
  stateIcons~remove(index)

  menu~delete(index)

return 0
```

29.3.16. removeAll

```
>>--removeAll------------------------------------><
```

Removes all images from the image list.

Arguments:

There are no arguments to this method.

Return value:

The method returns true on success, false otherwise.

Details:

Raises a syntax error if the image list is null.

Provides an interface to the **`ImageList_RemoveAll()`** API. The MSDN *documentation* documentation can provide more information on this method.

Example:

This example is a continuation of the fictious restaurant point-of-sale application. The waiter that wrote the application did not like to work past the normal closing time. The application has a feature that removes all the menu items as it gets near closing time to prevent customers from ordering something they do not have time to finish eating. (However, the waiter noticed that he often got huge tips from customers that had drinks after their meal.)

```
::method stopOrders private

  menu = self~newListView(IDC_LV_MENU)

  bigIcons = menu~getImageList(.Image~toID(LVSIL_NORMAL))
  bigIcons~removeAll

  smallIcons = menu~getImageList(.Image~toID(LVSIL_SMALL))
  smallIcons~removeAll

  stateIcons = menu~getImageList(.Image~toID(LVSIL_STATE))
  stateIcons~removeAll

  menu~deleteAll

  self~addAfterDinnerDrinks(menu)

return 0
```

29.4. ResourceImage Class

Resource Images represent modules that contain resources. These modules are binary executable files, which on Windows are for all practical purposes .exe and .dll files. Before an application can use a resource in a module, the resource must be loaded from the module into memory. Once in memory, the application needs a handle to the specific resource in order to work with it.

The ResourceImage class contains methods and function to assist in the loading of resources into memory and obtaining handles to the desired resources. ooDialog has always had, limited, access to resources in modules through the *ResDialog* class. The ResourceImage class expands that access, both improving it and making the access available in any ooDialog program, with or without ResDialog objects. The class supports the loading of, and obtaining handles to, resources from any binary executable file.

The class provides the *release* method to free up system resources if a module containing the resources is no longer needed. The general *remarks* concerning freeing the *Image* class are applicable here. The release() method will ignore requests to release the module when it is known absolutely that the module should not be released, such as in the case the module is the **`oodialog.dll`**. In other cases only the programmer can know if the module should or shouldn't be released. In those cases, the ResourceImage class will do as requested. As with the images that the .Image class represents, the system resources used by the module the ResourceImage class represents will be automatically cleaned up when the interpreter process ends. The *comments* about why would a programmer want to release resources in the **`Image`** class remarks are equally apropos here.

Note: The **`ResourceImage`** class is new in ooDialog as of ooRexx version 4.0.0. Its first implementation supports image resources and provides the basic framework upon which to expand in the future.

29.4.1. Method Table

Instances of the ResourceImage class implement the methods listed in the following table:

Table 29.4. ResourceImage Instance Methods

Method	Description
Class Methods	
new	Instantiates a new resource image that represents the executable module specified by the *fileOrDlg* argument.
Instance Methods	
getImage	Loads the specified image resource in the module and returns a new **`.Image`** object.
getImages	Loads a number of image resources from the module.
handle	Returns the Windows system handle for the resource module.
isNull	Determines if the resource image is valid.
release	Releases the resource module so that the operating system can reclaim the system resources.
systemErrorCode	Reflects any system error codes that are detected while working with an image object.

29.4.2. new (Class method)

```
>>--new(--fileOrDlg--)---------------------------><
```

Instantiates a new resource image that represents the executable module specified by *fileOrDlg*.

Arguments:

The single arguments is:

fileOrDlg [required]

This argument must be either a string file name or a dialog object. If *fileOrDlg* is a string, it specifies the file name of an executable module containing resources that the ooDialog program wants to access. If it is a dialog object in the current program, then it specifies that the module containing the resources is one of the modules already available to the program. That is, either the **`oodialog.dll`** module, or the module used for instantiating a *new* **`ResDialog`** object.

If the dialog object is a *ResDialog* it specifies the module used to instantiate the dialog object. If the dialog object is not a *ResDialog* then it specifies the oodialog.dll module, which is always loaded in any ooDialog program.

If *fileOrDlg* is a string file name, then the file name alone is sufficient. The operating system will search the path for the file as it normally does when a program is executed. If the module is not in the path then a full, or relative, file name is required.

Return value:

Returns a new *ResourceImage* object. This object may be null if an error occurred. Use the *isNull* method to check for this. Both the *.systemErrorCode* or the *systemErrorCode* method should contain an error code if the object is null.

Details:

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect arguments are detected.

Example:

This example instantiates a resource image using the oodialog.dll module and uses it to load the generally available icon resources from that module. The icons are then displayed in a list-view control.

```
::method initDialog
  expose list

  list  = self~newListView(IDC_LV_IMAGES)

  ids = .array~new()
  ids[1] = self~constDir[IDI_DLG_OODIALOG]
  ids[2] = self~constDir[IDI_DLG_APPICON]
  ids[3] = self~constDir[IDI_DLG_APPICON2]
  ids[4] = self~constDir[IDI_DLG_OOREXX]

  size = .Size~new(.SM~cxIcon, .SM~cyIcon)

  oodModule = .ResourceImage~new(self)
  icons = oodModule~getImages(ids, 'ICON', size)

  imageList = .ImageList~create(size, 'COLOR24 MASK', 4, 0)
  imageList~addImages(icons)

  list~setImageList(imageList, 'NORMAL'

  names = .array~new()
  names[1] = "IDI_DLG_OODIALOG"
  names[2] = "IDI_DLG_APPICON"
  names[3] = "IDI_DLG_APPICON2"
  names[4] = "IDI_DLG_OOREXX"

  do i = 1 to ids~items
    list~add(names[i] '('ids[i]')', i - 1)
  end
```

29.4.3. getImage

```
>>--getImage(--id--+--------+--+--------+--+---------+--)-------><
                   +-,-type-+  +-,-size-+  +-,-flags-+
```

Loads the specified image resource in the module and returns a new *Image* object.

Note: **`LoadImage()`** is the underlying Windows API used here. This method is very similar to the *getImage* method of the .Image class. The documentation for that method may provide additional insight.

Arguments:

The arguments are:

id
: The resource id of the image in the module.

  Note: At this time symbolic IDs are not supported for this argument. That restriction may be lifted in a future version of ooDialog.

type
: Specifies the type of the image: bitmap, icon, or cursor. You can use the *toID* method of the *Image* class to get the correct numeric value for one of the following symbols:

IMAGE_BITMAP	IMAGE_ICON
IMAGE_CURSOR	

  The default is IMAGE_BITMAP

size
: A *Size* object that specifies the size of the image. The default is a size of 0x0. Under most circumstances this indicates that the actual size of the image should be used. However, the MSDN *documentation* documentation for **`LoadImage()`** should be consulted for other meanings.

flags
: The load resource flags for the LoadImage() API. The flags are one or more of the following symbols. You can use the *toID* method of the *Image* class to get the correct numeric value for any of the following symbols. The *or* method of the *DlgUtil* class can be used to combine more than one of the symbols if needed.

LR_DEFAULTCOLOR	LR_CREATEDIBSECTION
LR_DEFAULTSIZE	LR_LOADFROMFILE
LR_LOADMAP3DCOLORS	LR_LOADTRANSPARENT
LR_MONOCHROME	LR_SHARED
LR_VGACOLOR	

  The default is LR_SHARED.

Return value:

This method returns an .Image object that represents the specified image resource in the module. The image may be null if an error occurred, for instance if no image resource was found.

Details:

Raises a syntax error if the resource image is null.

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect arguments are detected.

Provides an interface to the Win32 API: **`LoadImage()`**. Use the MSDN *documentation* documentation to get more information on the arguments to this method.

Example:

This example uses a ResDialog with a static control that displays a bitmap image. The image for the static control is obtained from the resource-only DLL used by the ResDialog.

```
::class SimpleDialog subclass ResDialog
...

::method initDialog
  ...
```

```
        resources = .ResourceImage~new("simpleImage.dll", self)
        image = resources~getImage(self~constDir[ID_BMP_IMAGE1])
        ...
        picture = self~newStatic(IDC_BMP_PICTURE)
        oldImage = picture~setImage(image)
```

29.4.4. getImages

```
>>--getImages(--ids,--+--------+--+--------+--+---------+--)----><
                      +-,-type-+  +-,-size-+  +-,-flags-+
```

Loads a number of image resources from the module. An array of resource IDs is used to specify which image resources. All the image resources must have the same characteristics. That is they must all be the same type and use the same flags. If the type is bitmap, and the flags include LR_DEFAULTSIZE, and the size is 0 x 0, then the size of the image will be the actual resource size. Otherwise, all the images also need to be the same size.

Note: `LoadImage()` is the underlying Windows API used here. This method is very similar to the *fromIDs* method of the .Image class. The documentation for that method can provide additional information.

Arguments:

The arguments are:

ids

An array of the resource IDs of the image resources in the module. The array can contain any number of IDs, but it can not be sparse. That is, each index of the array must contain a number. If an incorrect item is detected in the array, then a syntax error is raised and no images are returned.

If there is an error with an individual image resource, then the index in the array for that image is left empty. One way to check for this type of error is to compare the number of items in the returned array with the number of items in ID array.

type

Specifies the type of the image: bitmap, icon, or cursor. You can use the *toID* method of the *Image* class to get the correct numeric value for one of the following symbols:

IMAGE_BITMAP	IMAGE_ICON
IMAGE_CURSOR	

The default is IMAGE_BITMAP.

size

A *Size* object that specifies the size of the image. The default is a size of 0x0. Under most circumstances this indicates that the actual size of the image should be used. However, the MSDN *documentation* documentation should be consulted for other meanings.

flags

The load resource flags for the LoadImage() API. The flags are one or more of the following symbols. You can use the *toID* method of the *Image* class to get the correct numeric value for any of the following symbols. The *or* method of the *DlgUtil* class can be used to combine more than one of the symbols if needed.

LR_DEFAULTCOLOR	LR_CREATEDIBSECTION
LR_DEFAULTSIZE	LR_LOADFROMFILE
LR_LOADMAP3DCOLORS	LR_LOADTRANSPARENT

LR_MONOCHROME	LR_SHARED
LR_VGACOLOR	

The default is LR_SHARED.

Return value:

The return will be an array of .Image objects, one for each resource ID specified in the **`ids`** array, if there is no error. If there is an error loading the image of any specified ID, the corresponding index in the returned array will be empty.

If there is an error, both the *.systemErrorCode* and the *systemErrorCode* attribute of this resource image will be set (to the same code.) It is conceivable that an error could occur where the system does not set an error code, but unlikely.

Details:

Raises a syntax error if the resource image is null.

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect arguments are detected.

Provides an interface to the Win32 API: **`LoadImage()`**. You can use the MSDN *documentation* documentation to get more information on the interaction between the flags, size, and type arguments.

Example:

The following is a complete working dialog. Four bitmap images are used. Each time the user clicks the next button, the next image is displayed.

An array is constructed containing the resource IDs of the 4 images. A new resource image object is instantiated from the **`images.dll`** module. The getImages() method is then used to get an array of 4 .Image objects, using the array of resource IDs.

```
    /* Simple Dialog to display some images */

      dlg = .SimpleDialog~new( , 'simpleImage.h')
      if dlg~initCode = 0 then do
        dlg~createCenter(200, 247, "Simple Image Viewer", "", , "MS Shell Dlg 2", 8)
        dlg~execute("SHOWTOP", 14)
      end

    return 0
    -- End of entry point.

    ::requires "ooDialog.cls"

    ::class SimpleDialog subclass UserDialog

    ::method defineDialog

      self~createStatic(IDC_BMP_PICTURE, 10, 10, 20, 17, "BITMAP REALSIZEIMAGE")
      self~createStatic(IDC_ST_DESCRIPTION, 14, 190, 176, 20, "TEXT LEFT",
"Description")
      self~createPushButton(IDC_PB_NEXT, 10, 223, 50, 14, , "Next", onNext)
      self~createPushButton(IDOK, 140, 223, 50, 14, "DEFAULT", "Ok")

    ::method initDialog
      expose picture images description descriptions nextImage

      ids = .array~new
```

```
        ids[ 1] = self~constDir[ID_BMP_IMAGE1]
        ids[ 2] = self~constDir[ID_BMP_IMAGE2]
        ids[ 3] = self~constDir[ID_BMP_IMAGE3]
        ids[ 4] = self~constDir[ID_BMP_IMAGE4]

        descriptions = .array~new
        descriptions[1] = "Conrad, King of the Hill"
        descriptions[2] = "Berk, Squint Eye"
        descriptions[3] = "Cienna, Deer in the Headlights"
        descriptions[4] = "Vail, The Flower Child"

        picture = self~newStatic(IDC_BMP_PICTURE)
        description = self~newStatic(IDC_ST_DESCRIPTION)
        self~connectButtonEvent(IDC_PB_NEXT, "CLICKED", onNext)

        nextImage = 1
        module = .ResourceImage~new("images.dll")
        images = module~getImages(ids)

        self~onNext

      ::method onNext
        expose picture description images descriptions nextImage

        if nextImage > images~items then nextImage = 1

        picture~setImage(images[nextImage])

        description~setText(descriptions[nextImage])
        nextImage += 1
```

29.4.5. handle

```
>>--handle---------------------------------------><
```

Returns the Windows system handle for the resource module. It is an error to invoke this method if the resource image is null, or after the resource image has been released.

At this time, the handle is only useful for display.

Arguments:
There are no arguments.

Return value:
The return is the handle for the module the resource image represents.

Example:
This example ...

```

      ::method showHandles

        oodMod = .ResourceImage~new("oodialog.dll", self)
        resMod = .ResourceImage~new("simpleImage.dll", self)

        say 'The oodialog.dll module handle is:         ' oodMod~handle
        say 'The resource module for this ResDialog is:' resMod~handle


      /* The output, for a ResDialog, might be:
         The oodialog.dll module handle is:         0x009E0000
         The resource module for this ResDialog is: 0x00AF0000
      */
```

29.4.6. isNull

```
>>--isNull---------------------------------------><
```

Determines if the resource image is valid. **`isNull()`** will always return **`.true`** after the resource image has been released. It will also return **`.true`** if there was an error instantiating a new resource image object.

Arguments:

There are no arguments.

Return value:

Returns **`.true`** if the object is null, otherwise **`.false`**.

Example:

It is a good idea to check that a new resource image object is valid before using it.

```
      ::method initDialog
        ...
        resourceModule = .ResourceImage~new("simpleImage.dll", self)
        if \ resourceModule~isNull the do
          image = resources~getImage(self~constDir[ID_BMP_IMAGE1])
        else do
          -- Some error handling here.
        end
```

29.4.7. release

```
>>--release--------------------------------------><
```

Releases the resource module so that the operating system can reclaim the system resources. After the resource image has been released it can no longer be used. The programmer should not release the resource image while resources loaded from the module are still in use.

If the resource image represents the **`oodialog.dll`** module, or the module used for a ResDialog, then the resource image should never be released. The **`release()`** method will ignore release requests in those situations.

Arguments:

There are no arguments.

Return value:

This method return the system error code, which will be zero if there is no error, and non-zero if there is an error. The *.systemErrorCode* is also set to this value.

Details:

Raises a syntax error if the resource image is null.

Sets the *.systemErrorCode*.

Example:

In this example, the images for an image list are loaded from a resource module. The resource module is no longer needed after the image list is created. Since an image list makes a copy of the images added to it, once the image list has all the images added to it, the resource image can be released.

```
::method getImages private
  use strict arg resourceIDs, size, ilFlags

  imageList = .ImageList~create(size, ilFlags, 20, 10);

  module = .ResourceImage~new("zooImages.dll")
  images = module~getImages(resourceIDs)
  imageList~addImages(images)

  do i = 1 to images~items
    images[i]~release
  end
  module~release

  return imageList
```

29.4.8. systemErrorCode

```
>>--systemErrorCode------------------------------><
```

When a method for a resource image is invoked that sets the *.systemErrorCode*, the systemErrorCode attribute of the resource image object is also set. This can be useful if the programmer wants to check for an error code at some point when it is possible that .systemErrorCode has been reset.

This method can always be invoked, even if the resource image is null. It is very similar to the *systemErrorCode* method of the .Image class and the documentation for that method is generally applicable here.

Arguments:

This method does not take any arguments.

Return value:

The return is a system error code, which will usually be 0, no error.

Example:

This example uses the **`systemErrorCode`** attribute to get more information on a possible error, rather than the **`.systemErrorCode`**.

```
::method initDialog
  ...
  mod = .ResourceImage~new("simpleImage.dll", self)
  if mod~isNull the do
    self~writeToLog("Error with resource module simpleImage.dll")
    self~writeToLog("  RC: " mod~systemErrorCode)
    self~writeToLog("  msg:" SysGetErrorText(mod~systemErrorCode)
    return
  end
```

```
        image = resources~getImage(self~constDir[ID_BMP_IMAGE1])
        ...
```

User Input

Capturing user input is normally an important part of any program. In one sense, the ooDialog framework is almost completely about capturing user input as that is the primary role of a dialog.

The following table lists the general categories of user input in the Windows operating system, as described by the Windows documentation. Several of the categories have no current implmentation in ooDialog, or the implementation is scattered throughout the *dialog* and dialog *control* classes. These categories are mentioned here for completeness. Future versions of ooDialog may be enhanced to address those categories.

Table 30.1. ooDialog User Input Groups

User Input	Description
Common Dialog Library	Objects and routines used to easily gather user input in a standard way.
Keyboard Accelerators	Keyboard accelerators are keystrokes or combination of keystrokes that generate a command notification for a program.
Keyboard Input	The operating system generates keyboard input from the physical keyboard and provides the means for a program to receive and process that input. ooDialog has the **`Keyboard`** class to deal with functionality related to the keyboard. However, for historic reasons some of the keyboard related function is scattered throughout the dialog and dialog control classes.
Mouse Input	The mouse is an important, but optional, input device for programs. ooDialog programs deal with mouse input primarily through the **`Mouse`** class
Raw Input	There are many user input devices beside the traditional keyboard and mouse, such as a joystick or touchpad. The operating system provides a generalized means for programs to receive and process that input.

30.1. Common Dialog Library

The ooDialog framework provides a number of easy to use dialogs and routines that obtain user input in common scenarios. These dialogs and routines allow programmers to include simple graphical elements in their Rexx applications and collect user input in a standard way. The public routines can be called with a single line of code and the standard dialogs can usually be executed with only two lines of code. This does not require the programmer to do much set up and makes them simple to work with.

30.1.1. Class and Routine Table

The common dialogs and routines are listed the following table:

Table 30.2. Common Dialog Classes and Public Routines

Class / Routine	Description
AskDialog routine	Pops up a message box containing the specified question, a Yes, and a No push button.
CheckList class	A dialog with a group of one or more check boxes.
CheckList routine	A shortcut function to invoke the **`CheckList`** dialog.
ErrorDialog routine	Pops up a message box containing the specified error message, an Ok push button, and an error icon.

Class / Routine	Description
FileNameDialog routine	Causes a file selection dialog box to appear.
InfoDialog routine	Pops up a message box containing the specified text and an Ok push button.
InputBox class	Provides a simple dialog with a title, a message, an edit control, and the Ok and Cancel push buttons.
InputBox routine	A shortcut function to invoke an **`InputBox`** dialog as a function.
IntegerBox class	An **`InputBox`** dialog whose edit control allows only numerical data.
IntegerBox routine	A shortcut to invoke an **`IntegerBox`** dialog as a function.
ListChoice class	Provides a dialog with a single-selection list box, and the Ok and Cancel push buttons.
ListChoice routine	Provides a shortcut to invoke a **`ListChoice`** dialog.
MessageDialog routine	Pops up a message box with the options specified.
MultiInputBox class	A dialog that provides a title, a message, and one or more edit controls.
MultiInputBox routine	Provides a shortcut function to invoke a **`MultiInputBox`** dialog
MultiListChoice class	A **`ListChoice`** dialog that allows multi-item selection
MultiListChoice routine	Provides a shortcut to invoke a **`MultiListChoice`** dialog.
PasswordBox class	**`InputBox`** dialog with that uses a password edit control.
PasswordBox routine	A shortcut to invoke a **`PasswordBox`** dialog.
ProgressDialog class	A pre-built dialog with a progress bar that can be used to show the progress of a lengthy operation in an application
SingleSelection class	A dialog that has a group of radio buttons. The user can select only one item of the group.
SingleSelection routine	Provides a quick way to invoke a **`SingleSelection`** dialog.
TimedMessage class	Shows a message window for a defined duration.
TimedMessage routine	Provides a quick way to invoking a **`TimedMessage`** dialog.

30.1.2. Standard Dialog Classes

The standard dialog classes are:

- TimedMessage
- InputBox
- PasswordBox
- IntegerBox
- MultiInputBox
- ListChoice
- MultiListChoice
- CheckList

- SingleSelection

Preparation:

Standard dialogs are prepared by using the **`new`** method of the class, which in turn invokes the **`init`** method. The parameters for the **`new`** method are described for the **`init`** method of each class.

Execution:

The dialog is then run by using the **`execute`** method. For most of the standard dialogs **`execute`** returns the user's input if the OK button is clicked and the null string if the Cancel button is clicked to terminate the dialog. If there is more than one return value, execute returns the value 1 and stores the results in an attribute. Exceptions to the general rule are noted in the documentation for the individual dialog.

Functions:

Each standard dialog is also available as a callable function. These functions are described in the Public Routines *section* of this chapter.

30.1.2.1. CheckList Class

The CheckList class is a dialog with a group of one or more check boxes.

Execute:

Returns 1 (if OK was clicked). The check boxes selected by the user are marked in a stem variable with the value 1.

The method listed below is defined by this class.

30.1.2.1.1. init

```
>>-aCheckList~init(--message--,--title--,--labels.--,--datas.-->

>---+------------------------+--)----------------><
    +-,--+-----+--+--------+-+
         +-len-+  +-,--max-+
```

Arguments:

The arguments are:

message

A text string that is displayed on top of the check box group. Use it to give the user advice on what to do.

title

A text string for the dialog's title

labels.

A stem variable (do not forget the trailing period) containing all the labels for the check boxes

datas.

This argument is a stem variable (do not forget the trailing period) that you can use to preselect the check boxes. The first check box relates to stem item 101, the second to 102, and so forth. A value of 1 indicates selected, and a value of 0 indicates deselected.

For example, Datas.103=1 indicates that there is a check mark on the third box.

len

Determines the length of the check boxes and labels. If omitted, the size is calculated to fit the largest label.

max

The maximum number of check boxes in one column. If there are more check boxes than max - that is, labels. has more items than the value of max - this method continues with a new column.

Example:

The following example creates and shows a dialog with seven check boxes:

```
lst.1 = "Monday";   lst.2 = "Tuesday"; lst.3 = "Wednesday"
lst.4 = "Thursday"; lst.5 = "Friday";  lst.6 = "Saturday"
lst.7 = "Sunday"

do i = 101 to 107
   chk.i = 0
end

dlg = .CheckList~new("Please select a day!","Day of week",lst., chk.)
if dlg~execute = 1 then do
   say "You selected the following day(s): "
   do i = 101 to 107
      a = i-100
      if chk.i = 1 then say lst.a
   end
end
```

30.1.2.2. InputBox Class

The InputBox class provides a simple dialog with a title, a message, one entry line, an OK, and a Cancel push button.

Execute:

Returns the user's input

The methods listed below are defined by this class.

30.1.2.2.1. init

```
>>-aInputBox~init(--message--,--title--,--prevalue--,--len--)----><
```

The init method prepares the input dialog.

Arguments:

The arguments are:

message

A text string that is displayed in the dialog

title

A string that is displayed as the dialog's title in the title bar

prevalue
: A string to initialize the entry line. If you do not want to put any text in the entry line, just pass an empty string.

len
: The width of the entry line in dialog units

Example:

The following example shows an InputBox dialog with the title of *Input* and an entry line:

```
dlg = .InputBox~New("Please enter your email address", -
                    "Input", "user@host.domain", 150)
value = dlg~execute
say "You entered:" value
drop dlg
```

30.1.2.2.2. defineDialog

```
>>-aInputBox~DefineDialog--------------------------------------><
```

The defineDialog method is called by the create method of the parent class, which in turn is called at the very beginning of Execute. You do not have to call it. However, you may want to over-ride it in your subclass to add more dialog controls to the window. If you over-ride it, you have to forward the message to the parent class by using the keyword super.

30.1.2.2.3. AddLine

```
>>-aInputBox~AddLine(--x--,--y--,--l--)------------------------><
```

The AddLine method is used internally to add one entry line to the dialog.

30.1.2.2.4. execute

```
>>-aInputBox~Execute-------------------------------------------><
```

The execute method creates and shows the dialog. After termination, the value of the entry line is returned if the user clicks the OK button; a null string is returned if the user clicks on Cancel.

30.1.2.3. IntegerBox Class

The IntegerBox class is an *InputBox* dialog whose entry line allows only numerical data.

Subclass:
: This class is a subclass of the InputBox *InputBox* class

Execute:
: Returns the user's numeric input

The methods are the same as for the InputBox *InputBox* class, with the exception of Validate.

30.1.2.3.1. validate

```
>>-aIntegerBox~validate----------------------------------------><
```

The only method this subclass overrides is validate, which is one of the automatically called methods of PlainUserDialog. It is invoked by the OK method, which in turn is called in response to a push button event. This method checks whether or not the entry line contains a valid numerical value. If the value is invalid, a message window is displayed.

30.1.2.4. ListChoice Class

The ListChoice class provides a dialog with a list box, an OK, and a Cancel button. The selected item is returned if the OK push button is used to terminate the dialog.

Execute:
Returns the user's choice or a null string

The method listed below is defined by this class.

30.1.2.4.1. init

```
>>-aListChoice~init(--message--,--title--,--input.-------------->

>--+--------------------------------------------------+-------><
   +-,--+------+--+------------------------------+--)-+
        +-lenx-+  +-,--+------+--+--------------+-+
                       +-leny-+  +-,--preselect-+
```

The init method is used to initialize a newly created instance of this class.

Arguments:

The arguments are:

message
A text string that is displayed on top of the list box. Use it to give the user advice on what to do.

title
A text string for the dialog's title

input.
A stem variable (do not forget the trailing period) containing string values that are inserted into the list box

lenx, leny
The size of the list box in dialog units

preselect
Entry that is selected when list pops up

Example:

The following example creates a list choice dialog box where the user can select exactly one dessert:

```
lst.1 = "Cookies"; lst.2 = "Pie"; lst.3 = "Ice cream"; lst.4 = "Fruit"

dlg = .ListChoice~new("Select the dessert please","YourChoice",lst., , ,"Pie")
say "Your ListChoice data:" dlg~execute
```

30.1.2.5. MultiInputBox Class

The MultiInputBox class is a dialog that provides a title, a message, and one or more entry lines. After execution of this dialog you can access the values of the entry lines.

Execute:
: Returns 1 (if OK was clicked). The values entered by the user are stored in attributes matching the labels of the entry lines.

The methods are the same as for the InputBox *InputBox* class, with the exception of Init.

30.1.2.5.1. init

```
>>-aMultiInputBox~init(--message--,--title--,--labels.--,--datas.-->

>--+--------+--)---------------------------------><
   +-,--len-+
```

The init method is called automatically whenever a new instance of this class is created. It prepares the dialog.

Arguments:

The arguments are:

message
: A text string that is displayed on top of the entry lines. Use it to give the user advice on what to do.

title
: A text string that is displayed in the title bar.

labels.
: A stem variable containing strings that are used as labels on the left side of the entry lines. Labels.1 becomes the label for the first entry line, labels.2 for the second, and so forth.

datas.
: A stem variable (do not forget the trailing period) containing strings that are used to initialize the entry lines. The entries must start with 101 and continue in increments of 1.

len
: The length of the entry lines. All entry lines get the same length.

Example:

The following example creates a four-line input box. The data entered is stored in the object attributes that are displayed after dialog execution.

```
lab.1 = "First name"  ;     lab.2 = "Last name "
lab.3 = "Street and City" ; lab.4 = "Profession:"
```

```
addr.101 = "John" ; addr.102 = "Smith" ; addr.103 = ""
addr.104 = "Software Engineer"

dlg = .MultiInputBox~new("Please enter your address", ,
"Your Address", lab., addr.)
if dlg~execute = 1 then do
  say "The address is:"
  say dlg~firstname dlg~lastname
  say dlg~StreetandCity
  say dlg~Profession
end
```

30.1.2.6. MultiListChoice Class

The MultiListChoice class is an extension of the ListChoice class. It makes it possible for the user to select more than one line at a time. The execute method returns the selected items' indexes separated by blank spaces. The first item has index 1.

Subclass:
: This class is a subclass of the ListChoice *ListChoice*

Execute:
: Returns the index numbers of the entries selected

Preselect:
: Indexes of entries, separated by a blank, that are to be preselected. The first entry has index 1 and the rest are increments of one.

The methods are the same as for the ListChoice *ListChoice* class, except that execute returns the index numbers of the selected entries.

Example:
: The following example creates a multiple list choice box where the user can select multiple entries:

```
lst.1 = "Monday" ;   lst.2 = "Tuesday" ; lst.3 = "Wednesday"
lst.4 = "Thursday" ; lst.5 = "Friday" ;  lst.6 = "Saturday"
lst.7 = "Sunday"

dlg = .MultiListChoice~new("Select the days you are working this week", ,
                           "YourMultipleChoice",lst., , ,"2 5")
s = dlg~execute
if s <> "" then do while s \= ""
                  parse var s res s
                  say lst.res
               end
```

30.1.2.7. PasswordBox Class

The PasswordBox class is an InputBox dialog with an entry line that echoes the keys with asterisks (*) instead of characters.

Subclass:
: This class is a subclass of the InputBox *InputBox*

Execute:
: Returns the user's password

The methods are the same as for the InputBox *InputBox*, with the exception of AddLine.

30.1.2.7.1. AddLine

```
>>-aPasswordBox~AddLine(--x--,--y--,--l--)-------><
```

The AddLine overrides the same method of the parent class, InputBox, by using a password entry line instead of a simple entry line.

30.1.2.8. A Progress Dialog

ooDialog provides, as a standard dialog, an easy to use implementation of a progress dialog in the *ProgressDialog* class. This implementation includes two small helper classes for the **`ProgressDialog`** class, the *Interruptible* and the *Alerter* classes. These classes are documented in this section.

30.1.2.8.1. ProgressDialog Class

A **`ProgressDialog`** object is used to show a dialog containing a *ProgressBar*, an optional message, an optional status line, and an optional *Cancel* button. The dialog shows the progress a lengthy task is making to the user. The optional components of the dialog, the message text, the status line, et., can be used to convey more information concerning the progress, or the purpose of the dialog, to the user. The components can be used, or not used, according to the needs of the application.

The progress dialog internally creates and sizes the components of the dialog, and the dialog itself, depending on how it is configured. For instance, if the dialog is configured to skip the message and status components, and the progress is not cancelable by the user, the dialog will consist of a progress bar only. The width of the dialog will either be a calculated width, default width, or a configured width. The height of the dialog will be the proper height to accommodate the progress bar and other components, with margins. The layout of the dialog, from top to bottom, has a static control for the message at the top of the dialog, then a static control for the status line, then the progress bar, and finally a cancel button at the bottom right hand corner. The static message text, static status text, and progress bar controls are all the entire width of the dialog with an equal margin on each side.

By default, the width of the dialog is calculated based on the width of the message text. If there is no message, the width is calculated based on the width of a typical message. Or, the programmer can configure a width. The dialog maintains a minimum width that it will not be sized smaller than.

The dialog object has a number of attributes that are used configure the appearance and behaviour of the dialog. The attributes all have default values and the progress dialog can be simply used *as is*, or the programmer can change the values of any or all of the attributes. The methods of the **`ProgressDialog`** class are used to start, update, and close the dialog.

Normally, an application, shows the progress bar, starts its lengthy operation, and periodically, as it makes progress with the operation, it advances the progress bar position. When the operation finishes, the application ends the progress bar. With the **`ProgressDialog`** object, the programmer first configures the object by setting any attributes whose default values are not appropriate. The *begin* method shows the progress bar. As the operation progresses, the programmer advances the progress bar position using the *increase* method. The *updateStatus* and *updateStatusText* methods can also be used to update the status line as progress is made, if desired. Finally, the *complete* and *endNow* methods are used to end the dialog.

One important aspect of performing a lengthy operation is whether the user can, or can not, cancel the operation once it is begun. By default, the progress dialog can not be canceled by the user, and no cancel button is present in the dialog. If the user tries to end the dialog using the Escape key, the Close menu item in the system menu, or the close button in the title bar, a message box is put up explaining that the operation is not done yet and can not be canceled. The dialog is not allowed to close. This default behaviour is changed throught the *canCancel* attribute or through the *setInterruptible* method. The use of an *Interruptible* object gives the programmer an elegant way of handling and controlling a user's request to cancel the operation.

30.1.2.8.1.1. Method Table

The following table lists the class and instance methods of the **`ProgressDialog`** class:

Table 30.3. ProgressDialog Class Method Reference

Method	Description
Constant Methods	**constant Methods**
constants	Some constants defined by the **`ProgressDialog`** class may be of use to the programmer.
Class Methods	**Class Methods**
new	Instantiates a new **`ProgressDialog`** object.
Attribute Methods	**Attribute Methods**
absoluteWidth	Reflects whether the width of the dialog is calculated based on the width of the message text, or if the width is set to an absolute value by the application.
canCancel	The *canCancel* attribute specifies whether the user is allowed to cancel the progress dialog or not.
captionText	Reflects the text that will be used for the title of the progress dialog.
incrStep	Reflects the value of the step increment for the progress bar in the dilaog.
initialStatusText	Reflects the text that will be displayed on the status line in the progress dialog when the dialog is first shown.
marqueeMode	Reflects the status of the marquee mode of the progress bar.
marqueePause	Reflects the time in milliseconds between updates in the animation when in marquee mode.
msgHeight	Reflects the number of lines that the area for the message text will take up.
msgText	Reflects the text to be displayed in the message area of the progress dialog.
noMsg	Reflects whether the progress dialog will contain a message area or not.
noStatus	Reflects whether the progress dialog will have a status line or not.
rangeMax	Reflects the highest position the progress bar can be moved to.
rangeMin	Reflects the lowest position the progress bar can be moved to.
width	Reflects the, possible, width of the progress dialog.
Instance Methods	**Instance Methods**
begin	Begins the execution of the underlying Windows dialog.
complete	Moves the position of the progress bar to the end of the bar.
endNow	Ends the execution of the underlying progress dialog.

Method	Description
increase	The *increase* method moves the position of the progress bar by the value of the *incrStep* attribute, or by the specified amount if the *step* argument is specified.
setInterruptible	Passes an instantiated *Interruptible* object to the Rexx progress dialog object.
updateStatus	Updates the status line with the supplied text.
updateStatusText	Updates the status line using the exact text specified.

30.1.2.8.1.2. Constant Methods

The **`ProgressDialog`** class uses a number of *constant* values that are defined using the **`::constant`** directive. Some of these constants may be of use to the programmer when working with a progress dialog object. These constants are listed and documented in this section. The useful constants provided by the class are listed in the table below:

Table 30.4. Useful Progress Dialog Constant Reference

Constant Symbol	Description
IDC_PB_MAIN	The resource ID of the progress bar used in the dialog. This ID could be used to get the *ProgressBar* object in the dialog and manipulate it directly using the methods of the class.
IDC_ST_MSG	The resource ID of the static text control used for the message in the dialog. This ID could be used to get the *Static* object in the dialog and manipulate it directly using the methods of the class.
IDC_ST_STATUS	The resource ID of the static text control used for the status line in the dialog. This ID could be used to get the *Static* object in the dialog and manipulate it directly using the methods of the class.

30.1.2.8.1.3. new (Class Method)

```
>>--new(--+--------+--+-------+--+-------+--+-------+--+--------+--)-----------><
          +-capt-,-+  +-,-msg-+  +-,-min-+  +-,-max-+  +-,-step-+
```

Instantiates a new **`ProgressDialog`** object and sets its configuration attributes as specified.

Arguments:

The arguments are:

capt [optional]

Specifies the caption, the title, for the dialog. The *captionText* attribute is set to the value specified. If this argument is omitted, the *captionText* attribute is set to the default of *Application Processing ...*

msg [optional]

Specifies the text for a message to be displayed at the top of the dialog box. The *msgText* attribute is set to this value. If this argument is omitted, the *msgText* attribute is set to a default value of *Background processing is taking place. Please be patient.*

min [optional]

Sets the minimum value for the progress bar's *range*. The *rangeMin* attribute is set to this value. If this arugment is omitted, the *rangeMin* attribute is set to 0.

max [optional]

Sets the maximum value for the progress bar's *range*. The *rangeMax* attribute is set to this value. If this arugment is omitted, the *rangeMax* attribute is set to 100.

step [optional]

Specifies the *step* increment for the progress bar. The *incrStep* attribute is set to this value. If this argument is omitted, the *stepIncr* attribute is set to a default value of 1.

Return value:

Returns a newly instantiated **`ProgressDialog`** object.

Remarks:

The exact behaviour and appearance of the progress dialog is configured by setting the values of its attributes. Only a few of the attribute's values can be set by arguments to the *new* method. The other attriubtes are all set to default values when the object is instantiated. The documentation for the individual attributes specifies what the default value for the attribute is.

Details:

Raises syntax errors when incorrect usage is detected.

Example:

This example instantiates a progress dialog, specifying a title and message for the dialog. It changes the range from 0 to 100, to 0 to 200, and changes the step increment value to 4:

```
title = 'Acme Widgets - Business Contacts'
msg = 'Creating and inserting the contact information. This will take some time.'

pbDlg = .ProgressDialog~new(title, msg, , 200, 4)
...
```

30.1.2.8.1.4. absoluteWidth (Attribute)

```
>>--absoluteWidth--------------------------------><

>>--absoluteWidth = trueOrFalse------------------><
```

Reflects whether the width of the dialog is calculated based on the width of the message text, or if the width is set to an absolute value by the application. This attribute is set to true or false. By default, the width of the dialog is calculated based on the width of the message text, or the width of the default message if the *msgText* attribute is not set.

However, the programmer can change this behavior by setting *absoluteWidth* to true and setting the *width* attriubte to a desired width.

Remarks:

When *absoluteWidth* is false, the value of the *width* attribute is always ignored. False is the default value for *absoluteWidth*. The programmer needs to set both attributes to force a desired width for the progress bar dialog.

Note that the progress dialog can not be made smaller in width than a minimum width enforced by the class. This minimum width is the width of 2 buttons plus the margins and the space between 2 buttons.

Example:

In this example the programmer wants the progress dialog to be wider than that calculated from the message text and wider than the enforced minimum:

```
title = 'Acme Widgets - Business Contacts'
msg = 'Inserting information.'

pbDlg = .ProgressDialog~new(title, msg, , 200, 4)

pbDlg~absoluteWidth = .true
pbDlg~width = 196
...
```

30.1.2.8.1.5. canCancel (Attribute)

```
>>--canCancel------------------------------------><

>>--canCancel = trueOrFalse----------------------><
```

The *canCancel* attribute specifies whether the user is allowed to cancel the progress dialog or not. By default, the dialog can not be canceled and the value of this attriubte is false. If this attribute is true, then a Cancel button is added to the dialog and the user is allowed to close, to cancel, the dialog.

Remarks:

If the programmer configures the dialog such that it can be canceled, then this implies to the user that closing the dialog halts the lengthy operation in progress. It is the programmer's responsibility to handle that. The *Interruptible* class and the *setInterruptible* method provide some tools to allow the programmer to handle canceling the dialog in a graceful manner.

If the *setInterruptible* method is used to assign an **`Interruptible`** object, the *canCancel* attribute is automatically set to true.

30.1.2.8.1.6. captionText (Attribute)

```
>>--captionText----------------------------------><

>>--captionText = title--------------------------><
```

Reflects the text that will be used for the title of the progress dialog. By default this attribute is set to *Application Processing ...*

Remarks:

The *captionText* attribute can also be set by using the *capt* argument in the *new* method.

Example:

This example instantiates a new progress dialog object and then configures its title and message:

```
pbDlg = .ProgressDialog~new

pbDlg~captionText = 'Acme Widgets - Business Contacts'
pbDlg~msgText = 'Creating and inserting the contact information. ' || -
                'This will take some time.'

...
```

30.1.2.8.1.7. incrStep (Attribute)

```
>>--incrStep-------------------------------------><

>>--incrStep = increment-------------------------><
```

Reflects the value of the step increment for the progress bar in the dilaog. The step increment is the amount by which the progress bar advances its current position when the *increase* method is called without an increment value.

Remarks:

By default the step increment for the progress bar is set to 1. Each time the *increase* method is invoked with no arguments, the position in the progress bar is advanced by the value of the step increment. If the progress bar has a range of 0 - 100 and the step increment is 4, invoking the *increase* method 25 times would move the position in the progress bar to the end.

Example:

This example instantiates a new progress dialog and then configures its step increment to 4.

```
title = 'Acme Widgets - Business Contacts'
msg = 'Inserting information.'

pbDlg = .ProgressDialog~new(title, msg)

pbDlg~incrStep = 4
...
```

30.1.2.8.1.8. initialStatusText (Attribute)

```
>>--initialStatusText----------------------------><

>>--initialStatusText = text---------------------><
```

Reflects the text that will be displayed on the status line in the progress dialog when the dialog is first shown. The status line is displayed just above the progress bar in the dialog. By default, if the *noStatus* attribute is not set to true, the status line will display *Status:*. The programmer can change the value of the *initialStatusText* so that some other text is displayed when the dialog starts. Perhaps the empty string.

Example:

This example configures the progress dialog to have a status line, but to display no text for the status when the dialog starts. Then, as the operation of creating rows proceeds, the application periodically updates the status line with the number of rows created.

```
pbDlg = .ProgressDialog~new
pbDlg~initialStatusText = ''
...
pbDlg~begin
...
pbDlg~updateStatus(i 'full rows created')
```

30.1.2.8.1.9. marqueeMode (Attribute)

```
>>--marqueeMode----------------------------------><

>>--marqueeMode = onOrOff------------------------><
```

Reflects the status of the marquee mode of the progress bar. By default, the attribute is false, the progress bar will not use marquee mode. Setting this attributed to true has the progress bar use marquee mode.

Remarks:

Marquee mode animates a progress bar in a way that shows activity but does not indicate what proportion of the task is complete. The highlighted part of the progress bar moves repeatedly along the length of the bar. The animation of the progress bar can be started and stopped. The speed of the animation can also be controlled. Marquee progress bars do not have a range or position.

The initial speed of the animation is controlled through the *marqueePause* attribute. Starting, stopping the animation, or changing its speed after the progress dialog is started can be done by using the *constant* ID of the progress bar to retrieve the *ProgressBar* object and using the *setMarquee* method.

Example:

This example configures a progress dialog to use marquee mode with an initial pause time of 50 milliseconds. At some later point during the operation, the animation is turned off. At a later point following that, the animation is turned back on with a pause time of 25 milliseconds:

```
progressDlg = .ProgressDialog~new
progressDlg~marqueeMode = .true
progressDlg~marqueePause = 50

...

pb = progressDlg~newProgressBar(progressDlg~IDC_PB_MAIN)

-- Turn the animation off
pb~setMarquee(.false)

...

-- Turn the animation back on, but faster
pb~setMarquee(.true, 25)
```

30.1.2.8.1.10. marqueePause (Attribute)

```
>>--marqueePause---------------------------------><

>>--marqueePause = milliseconds------------------><
```

Reflects the time in milliseconds between updates in the animation when in marquee mode. The default for this attriubte is 100 milliseconds. The value of the attribute is ignored unless the progress bar is put in marquee *mode*.

Example:

This example configures the progress dialog to use marquee mode for the progress bar and sets the pause time ot 50 milliseconds instead of the default 100.

```
progressDlg = .ProgressDialog~new
```

```
  progressDlg~marqueeMode = .true
  progressDlg~marqueePause = 50
```

30.1.2.8.1.11. msgHeight (Attribute)

```
>>--msgHeight------------------------------------><

>>--msgHeight = lines----------------------------><
```

Reflects the number of lines that the area for the message text will take up. By default, *msgHeight* is set to 1 and the width of the dialog is calculated by calculating the width of the message. The height of 1 line is calculated from the height of the message. This doesn't work well for very long messages. The programmer may want set the message area height to 2 or more lines to end up with a more balanced dialog.

Remarks:

If the *msgHeight* attribute is greater than 1, the width of the dialog is calculated using the width of the default message *text*. The height of 1 line is always calculated using some text, the default message, the assigned message, etc.. The height calculation is not effected by what the exact text is as long as the text has a few words in it.

The height of the static control for the message is *always* set to *msgHeight* times the calculated height of a single line. The static control for the message will automatically wrap the text at word breaks. The control also honors new line characters if they are embedded in the text.

No attempt is made during the height and width calculations to try and determine if the assigned message text will fit in the static control or not if the *msgHeight* is set higher than 1. This means the programmer can increase the over all height of the dialog by increasing the message height, even if the height does not need to be increased to accommodate the message text.

Example:

In this example, the application sometimes adds some extra text to the message. Knowing that the progress dialog will be very short and very long without some change, the message height is set to 5 lines to produce a balanced dialog. Note the insertion of two new lines in the message text to set off the added text:

```
pbDlg = .ProgressDialog~new
pbDlg~msgText = 'Creating and inserting the contact information. ' || -
                'This will take some time.'

if self~itemCount > 30000 then do
  extra = .endOfLine~copies(2) || -
          'It is possible that inserting' self~itemCount 'items'   || -
          ' will exhaust your system resources.'

  pbDlg~msgText ||= extra
  pbDlg~msgHeight = 5
end
```

30.1.2.8.1.12. msgText (Attribute)

```
>>--msgText--------------------------------------><
```

```
>>--msgText = text------------------------------><
```

Reflects the text to be displayed in the message area of the progress dialog.

Remarks:

The default value for the message text is *Background processing is taking place. Please be patient.* The message text is displayed at the top of the progress dialog. To remove the message entirely, use the *noMsg* attribute. To control the number of lines used for the message, use the *msgHeight* attribute.

Example:

This example changes the default message to one appropriate for the specific application:

```
pbDlg = .ProgressDialog~new
pbDlg~msgText = 'Creating and inserting the contact information. ' || -
                'This will take some time.'
```

30.1.2.8.1.13. noMsg (Attribute)

```
>>--noMsg---------------------------------------><

>>--noMsg = trueOrFalse--------------------------><
```

Reflects whether the progress dialog will contain a message area or not. The default is false, the dialog will contain a message area. If the *noMsg* attriubte is set to true, the message area is removed from the progress dialog, and the *msgText* attribute is ignored.

Remarks:

When *noMsg* is true, by default, the width of the dialog is calculated using the width of the default message text. However, the setting of the *absoluteWidth* attribute can also have an effect on the width calculation.

Example:

This example configures a progress dialog so that it has no status line area and no message text area.

```
progressDlg = .ProgressDialog~new
progressDlg~noMsg = .true
progressDlg~noStatus = .true
```

30.1.2.8.1.14. noStatus (Attribute)

```
>>--noStatus------------------------------------><

>>--noStatus = trueOrFalse-----------------------><
```

Reflects whether the progress dialog will have a status line or not. If *noStatus* is true, no area for the status line is included in the progress dialog. The default is false, the dialog includes a status line.

Remarks:

If the status line is desired, but the default status text is not appropriate, the *initialStatusText* attribute can be set to the empty string, or to a string appropriate for the application.

Example:

This example configures a progress dialog so that it has no message text area and no status line area.

```
progressDlg = .ProgressDialog~new
progressDlg~noStatus = .true
progressDlg~noMsg = .true
```

30.1.2.8.1.15. rangeMax (Attribute)

```
>>--rangeMax-------------------------------------><

>>--rangeMax = max-------------------------------><
```

Reflects the highest position the progress bar can be moved to.

Remarks:

The *rangeMax* attribute can also be set using the *max* argument in the *new* method.

30.1.2.8.1.16. rangeMin (Attribute)

```
>>--rangeMin-------------------------------------><

>>--rangeMin = min-------------------------------><
```

Reflects the lowest position the progress bar can be moved to.

Remarks:

The *rangeMin* attribute can also be set using the *min* argument in the *new* method.

30.1.2.8.1.17. width (Attribute)

```
>>--width----------------------------------------><

>>--width = dlgUnits-----------------------------><
```

Reflects the, possible, width of the progress dialog. The *width* attribute is ignored unless the *absoluteWidth* attribute is set to true. By default *absoluteWidth* is false. If *absoluteWidth* is true, then the width of the progress bar, in dialog units, is set to the value of the *width* attribute.

Remarks:

When the progress dialog object is instantiated, the *width* attribute is set to the minimum width the dialog will be set to. This is done simply to ensure the attribute has a numerical value. It is expected that the if the application sets the *absoluteWidth* attribute to true, the application will also set the *width* attribute to some specific value.

Example:

This example configures a progress dialog to not have a message or a status line. Rather than allow the progress dialog to calculate a width, the application sets the width to a specific value:

```
title = 'Acme Widgets - Business Contacts'
```

```
    pbDlg = .ProgressDialog~new(title)

    pbDlg~noMsg
    pbDlg~noStatus
    pbDlg~absoluteWidth = .true
    pbDlg~width = 192
    ...
```

30.1.2.8.1.18. begin

```
>>--begin----------------------------------------><
```

Begins the execution of the underlying Windows dialog. The *begin* method is similar in concept to the *popup* method of the dialog object.

Arguments:

This method has no arguments.

Return value:

Returns 0 on success and 1 for some unexpected error.

Remarks:

The *begin* method starts the dialog and returns immediately. If there is some error creating the underlying dialog, a message box is put up explaining the problem to the user. This is highly unlikely to happen.

Example:

This example configures a progress dialog and then uses the *begin* method to start it:

```
    pbDlg = .ProgressDialog~new
    pbDlg~msgText = 'Creating and inserting the contact information. ' || -
                    'This will take some time.'

    a = .Alerter~new
    pbDlg~setInterruptible(a)
    pbDlg~begin
    pbDlg~updateStatus('0 full rows created')
```

30.1.2.8.1.19. complete

```
>>--complete-------------------------------------><
```

Moves the position of the progress bar to the end of the bar.

Arguments:

There are no arguments for this method.

Return value:

There is no return from this method.

Remarks:

This method is intended to be invoked when the lengthy operation is finished. Moving the position of the progress bar to the end gives the user a visual clue that the operation has ended. It is not necessary to use this method to finish the dialog, it is a convenience method. The *endNow* method actually ends the dialog, and is required to end the dialog when it has not been canceled by the user.

Example:

This example uses the *complete* method to move the position of the progress bar to the end when it drops out of the processing loop. It then updates the status line and pauses a short time before actually ending the dialog. This gives the user a visual clue that the operation completed successfully:

```
do i = 1 to rows~items
    ...
end

pbDlg~complete
pbDlg~updateStatus('finished inserting' rows~items 'full rows')
r = SysSleep(1.5)
pbDlg~endNow
```

30.1.2.8.1.20. endNow

```
>>--endNow---------------------------------------><
```

Ends the execution of the underlying progress dialog.

Arguments:

This method does not have any arguments.

Return value:

Returns true.

Remarks:

If the progress bar is cancelable and the user cancels the dialog, that will of course close the dialog. If the dialog is not cancelable, then the user can not close the dialog and the programmer must end the dialog using the *endNow* method. Good practice would probably be to close the dialog for the user when the lengthy operation is ended.

Example:

This example supplies the user with a visual clue that the operation completed successfully when it drops out of the processing loop. It then ends the dialog using the *endNow* method.

```
do i = 1 to rows~items
    ...
end

pbDlg~complete
pbDlg~updateStatus('finished inserting' rows~items 'full rows')
r = SysSleep(1.5)
pbDlg~endNow
```

30.1.2.8.1.21. increase

```
>>--increase(--+--------+--)---------------------><
               +--step--+
```

The *increase* method moves the position of the progress bar by the value of the *incrStep* attribute, or by the specified amount if the *step* argument is specified.

Arguments:

The single argument is:

step [optional]

A specific amount to move the progress bar by. If this argument is omitted the value of the *incrStep* attribute is used.

Return value:

This method does not return a value.

Remarks:

The *increase* method is exactly equivalent to the *step* method of the *ProgressBar* class.

If the progress bar is in marquee mode, or the progress dialog was not instantiated correctly, this method does nothing. If the underlying dialog has not been created yet, this method will wait until it has been created.

Example:

This example immediately invokes the *increase* method after it starts the dialog. If the underlying Windows dialog has not been fully created, the *increase* method will wait until it has been created:

```
pbDlg = .ProgressDialog~new
pbDlg~msgText = 'Creating and inserting the contact information. ' || -
                'This will take some time.'

a = .Alerter~new
pbDlg~setInterruptible(a)
pbDlg~begin
pbDlg~increase
```

30.1.2.8.1.22. setInterruptible

```
>>--setInterruptible(--interruptibleObject--)----><
```

Passes an instantiated *Interruptible* object to the Rexx progress dialog object.

Arguments:

The single argument is:

interruptibleObject [required]

The interruptible object the progress dialog should use.

Return value:

Returns 0 on success, 1 on error.

Remarks:

An **`Interruptible`** object provides a generic way for the progress dialog to handle the user canceling the dialog. When the *setInterruptible* method is invoked, the *canCancel* attribute is automatically set to true.

If the user cancels the dialog, the progress dialog will first invoke the *interruptible* method to determine if the lengthy operation in progress can be interrupted at that point. If the return is false, the progress dialog informs the user, through a message dialog, that the operation can not be canceled and does not close the dialog. If the return is true, then the progress dialog invokes the *interrupt* method to notify the *interruptible* object that the user canceled the operation, and ends the dialog.

Using those methods of the *interruptible* object, the programmer can manage how and when the user can cancel the operation and react correctly to a cancellation.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example instantiates an *Alerter* object, which is a concrete implementation of the **`Interruptible`** class, and passes the object to the progress dialog. This allows the user to cancel the dialog. The **`Alerter`** class provides the *isCanceled* attribute. By checking the *isCanceled* attribute within the processing loop, the application can determine if the user has canceled the operation and end the processing loop:

```
pbDlg = .ProgressDialog~new
pbDlg~msgText = 'Creating and inserting the contact information. ' || -
                'This will take some time.'

a = .Alerter~new
pbDlg~setInterruptible(a)
pbDlg~begin

do i = 1 to count
    -- do stuff

    if /* some condition */ then do
          pbDlg~increase
    end

    if a~isCanceled then do
          pbDlg~updateStatus('canceled after ...' )
          r = SysSleep(1.5)
          pbDlg~endNow
          leave
    end
end
```

30.1.2.8.1.23. updateStatus

```
>>--updateStatus(--text--)-----------------------><
```

Updates the status line with the supplied text. The status line will always begin with the string: *Status:* and continue with the supplied text.

Arguments:

The single argument is:

text [required]
: The text to append after the *status :* string.

Return value:

There is no return from this method.

Remarks:

To change the status line text without having the *Status:* string prepended use the updateStatusText method.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example creates a progress dialog and enters a processing loop. Every 100 iterations, it updates the status line using the *updateStatus* method. When i equals 200, the status line will read: *Status: 200 records inserted.*

```
pbDlg = .ProgressDialog~new
pbDlg~msgText = 'Creating and inserting the contact information. ' || -
                'This will take some time.'

pbDlg~begin

do i = 1 to count
    -- do stuff

    if i // 100 = 0 then do
          pbDlg~increase
          pbDlg~updateStatus(i 'records inserted.')
    end
    ...
end
```

30.1.2.8.1.24. updateStatusText

```
>>--updateStatusText(--text--)-------------------><
```

Updates the status line using the exact text specified.

Arguments:

The single argument is:

text [required]

The text to be displayed on the status line.

Return value:

There is no return from this method.

Remarks:

The similar, updateStatus method, always starts the status line with *Status:* . This is a convenience, sometimes. This method is provided for situations where the *Status:* prolog is undesired.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example creates a progress dialog and enters a processing loop. Every 100 iterations, it updates the status line using the *updateStatusText* method. When i equals 200, the status line will read: *Have now inserted the contact information for 200 contacts ...*

```
  pbDlg = .ProgressDialog~new
  pbDlg~msgText = 'Creating and inserting the contact information. ' || -
                  'This will take some time.'

  pbDlg~begin

  do i = 1 to count
      -- do stuff

      if i // 100 = 0 then do
            pbDlg~increase
            pbDlg~updateStatus('Have now inserted the contact information for' i
 'contacts ...')
      end
      ...
  end
```

30.1.2.8.2. Alerter Class

The **`Alerter`** class is a simple implementation of the *Interruptible* interface that can be used with a *ProgressDialog* object to notify the application that the user has canceled the progress dialog.

An **`Alerter`** object is intended to be used when the progress dialog can always be canceled by the user, but the application needs to know if and when the user cancels.

30.1.2.8.2.1. Method Table

The following table lists the class and instance methods of the **`Alerter`** class:

Table 30.5. Alerter Class Method Reference

Method	Description
Class Methods	**Class Methods**
new	Instantiates a new **`Alerter`** object.
Attribute Methods	**Attribute Methods**
isCanceled	Reflects the state of the internal canceled flag.
Instance Methods	**Instance Methods**
interrupt	Signals this object that it is being interrupted.
interruptible	Determines if this object is interruptible at this point in time. By definition an **`Alerter`** object is always interruptible.

30.1.2.8.2.2. new (Class Method)

```
>>--new------------------------------------------><
```

Instantiates a new **`Alerter`** object.

Arguments:

There are no arguments for this method.

Return value:

Returns a newly instantiated **`Alerter`** object.

Remarks:

An *Alerter* object provides a concrete implementation of the two abstract methods of the **`Interruptible`** interface. The object sets the *isCanceled* attributed to false on instantiation. if the *interrupt* method is invoked, the attribute is set to true. This allows the application to test the value of the *isCanceled* attribute to determine if the user has canceled the progress dialog.

Example:

This example shows an **`Alerter`** object being used with a progress dialog. Within a processing loop, the application periodically checks to see if the user has canceled the operation by checking the *isCanceled* attribute of the alerter:

```
pbDlg = .ProgressDialog~new
pbDlg~msgText = 'Creating and inserting the contact information. ' || -
                'This will take some time.'

a = .Alerter~new
pbDlg~setInterruptible(a)
pbDlg~begin

do i = 1 to count
    -- do stuff

    if /* some condition */ then do
          pbDlg~increase
    end

    if a~isCanceled then do
          pbDlg~updateStatus('canceled after ...' )
          r = SysSleep(1.5)
          pbDlg~endNow
          leave
    end
end
```

30.1.2.8.2.3. isCanceled (Attribute)

```
>>--isCanceled-----------------------------------><
```

Reflects the state of the internal canceled flag. The flag is false if the *interrupt* method of this object has not been invoked and true if it has been invoked.

Remarks:

The *isCanceled* attribute is a *get* only attribute. The programmer can not set the value of this attribute. It is set internally by the object and can not be changed.

30.1.2.8.2.4. interrupt

```
>>--interrupt------------------------------------><
```

Signals this object that it is being interrupted.

Arguments:

There are no arguments to this method.

Return value:

This method does not return any value.

Remarks:

The *alerter* object provides a concrete implmentation of the *Interruptible* interface that can be used as the arguemnt to the *setInterruptible* method of the *ProgressDialog*. It is intended to be used when the progress dialog can always be canceled and provides a notification to the application if the progress dialog is canceled.

When the *interrupt* method is invoked, the *isCanceled* attribute is set to true. When used with the progress dialog object, the progress dialog invokes this method if the user cancels the dialog. This allows the programmer to determine if the user has canceled the dialog by checking the *isCanceled* attribute.

30.1.2.8.2.5. interruptible

```
>>--interruptible--------------------------------><
```

Determines if this object is interruptible at this point in time. By definition an **`Alerter`** object is always interruptible.

Arguments:

There are no arguments for this method.

Return value:

This method always returns true.

Remarks:

The *alerter* object provides a concrete implmentation of the *Interruptible* interface that can be used as the arguemnt to the *setInterruptible* method of the *ProgressDialog*. It is intended to be used when the progress dialog can always be canceled and provides a notification to the application if the progress dialog is canceled. When used with the progress dialog object, the progress dialog invokes this method if the user cancels, to determine if it is okay to cancel, to close, the dialog. This method always returns true for that invocation.

30.1.2.8.3. Interruptible Mixin Class

The **`Interruptible`** class defines an interface for an object that can be *interrupted*, but possibly only interrupted at certain times. It does this by defining two abstract methods. Users of the **`Interruptible`** class are expected to inherit this mixin class and provide concrete implementations of both abstract methods.

The **`Interruptible`** class is a helper class for the *ProgressDialog* class. Only *interruptible* objects can be used as an argument for the *setInterruptible* method. The progress dialog will invoke both methods of the **`Interruptible`** object, so concrete implementations for both methods must be provided to prevent syntax conditions from being raised.

30.1.2.8.3.1. Method Table

The following table lists the abstract instance methods of the **`Interruptible`** mixin class:

Table 30.6. Interruptible Class Method Reference

Method	Description
Instance Methods	**Instance Methods**
interrupt	Signals this object that it is being interrupted.
interruptible	Determines if this object is interruptible at this point in time.

30.1.2.8.3.2. interrupt (abstract)

```
>>--interrupt------------------------------------><
```

Signals this object that it is being interrupted.

Arguments:

There are no arguments to this method.

Return value:

This method does not return any value

Remarks:

The contract for this abstract method is that it will not receive any arguments and will not return any value.

30.1.2.8.3.3. interruptible (abstract)

```
>>--interruptible--------------------------------><
```

Determines if this object is interruptible at this point in time.

Arguments:

There are no arguments for this method.

Return value:

Returns true if this object can be interrupted now. Returns false if this object should not be interrupted now, but can possibly be interrupted at a later time.

Remarks:

The contract for this abstract method is that it will not be passed any arguments and will always return either true or false.

30.1.2.9. SingleSelection Class

The **`SingleSelection`** class presents a dialog that has a group of *radio* buttons. The user can select only one item of the group, or cancel the dialog. The programmer can specify how many radio buttons are in the group, if they should be placed in one, or several, columns. The ooDialog framework internally handles all the layout of the controls of the dialog. This makes the dialog class very easy to use.

The basic steps to user the **`SingleSelection`** are to instantiate the dialog using the *new* method, run the dialog using the *execute* method, and do something with the user's response which is returned from the *execute* method.

30.1.2.9.1. new

```
>>-new(--msg--,--title--,--labels.--,--+----------+--+-----------+--+-------+--)--><
                                       +-,-preSel-+  +-,-rbWidth-+  +-,-max-+
```

Instantiates a new **`SingleSelection`** object.

Arguments:

The arguments are:

msg [required]
: A text string that is displayed above the radio button group. Typically, the text would be used to give the user the purpose of the radio buttons.

title [required]
: A title, the caption bar text, for the dialog

labels. [required]
: A stem with whole number numeric indexes, one through the number of radio buttons desired. The value for each index is the label for the corresponding radio button. For instance **`labels.2`** would be set to the label for the second radio button.

preSel [optional]
: This argument can be used to preselect, (check,) one radio button. A value of 1 checks the first radio button, a value of 2 checks the second, and so on. If this argument is omitted, the first radio button is checked.

rbWidth [optiona]
: Specifies the width, in dialog units, of the radio button controls. If omitted, the width is calculated so that the longest label will not be clipped. If used, all radio buttons are set to this width without regard as to whether the label will be clipped or not.

max [optional]
: The maximum number of radio buttons in one column. If there are more radio buttons than *max*, that is, if the *labels.* argument has more indexes than the value of *max*, the radio buttons are positioned in columns. Each column will have *max* radio buttons and ther will be as many columns as needed to display all the radio buttons.

Example:

The following example instantiates and executes a dialog that contains a two-column radio button group. The fifth radio button (the button with the label May) is preselected.

```
mon.1 = "January" ; mon.2 = "February" ; mon.3 = "March"
mon.4 = "April"   ; mon.5 = "May"      ; mon.6 = "June"
mon.7 = "July"    ; mon.8 = "August"   ; mon.9 = "September"
mon.10= "October" ; mon.11= "November" ; mon.12= "December"

dlg = .SingleSelection~new("Please select a month:", ,
                           "Single Selection", mon., 5, , 6)
s = dlg~execute
say  "You Selected the month:" mon.s
```

30.1.2.9.2. execute

```
>>--execute--------------------------------------><
```

Executes the dialog and returns the user's selection or the empty string if the user cancels the dialog.

Arguments:

There are no arguments to this method

Return value:

If the user cancels the dialog, the empty string is returned. If the user closes the dialog with *Ok* the return is the number of the selected radio button. 1 for the first radio button, 2 for the second radio button, 8 for the eighth radion button, etc., etc..

Remarks:

When the *execute* method is invoked, the ooDialog framework, internally, does a number of calculations to determine the optimal size of the dialog controls and dialog. The framework then positions all the dialog controls such that none of the radio button labels, message, and title of the dialog are clipped.

The one exception to this is if the programmer specifies the width of the radio button controls using the optional *rbWidth* argument in the new method. In that case the width of the radio buttons is always set to what the programmer has specified. It is possible that the radio buttons will be clipped, but that is now the responsibility of the programmer.

Example:

This example instantiates a single selection dialog

```
rex.1 = "Regina" ; rex.2 = "ooRexx" ; rex.3 = "Reginald" ; rex.4 = "Personal Rexx"

d = .SingleSelection~new("Pick your favorite Rexx", "Rexx is King", rex., 2, , 2)
index = d~execute

if index == "" then say "You failed to pick a Rexx interpreter."
else say "The Rexx you picked:" rex.index

::requires 'ooDialog.cls'

/* Output will most likely be:

The Rexx you picked: ooRexx

*/
```

30.1.2.10. TimedMessage Class

The TimeMessage class shows a message window for a defined duration.

The methods listed below are defined by this class.

30.1.2.10.1. init

```
>>-aTimedMessage~init(--msg-,-title-,-sleeping--+-------------+-+-----+--)----><
```

```
                                                +-,-earlyReply-+ +-,-p-+
```

The init method prepares the dialog.

Arguments:

The arguments are:

message

A string that is displayed inside the window as a message. The length of the message determines the horizontal size of all standard dialogs.

title

A string that is displayed as the window title in the title bar of the dialog

sleeping

A number that determines how long (in milliseconds) the window is shown. If this number is 0 or greater, the message window is shown for that time and automatically dismisses itself. The programmer need take no further action.

If the number is less than 0 then the message window is shown indefinitely. The **`execute`** method returns immediately. In this case, the programmer needs to dismiss the window manually. This is done by invoking the **ok** method. See **Example 2** below.

earlyReply

The optional early reply argument defaults to false. If used and if true, the **`execute`** method of the dialog returns immediately. This allows the code at the point where **`execute`** was invoked to continue. The message windows continues to display for its specified duration and then dismisses itself automatically.

p [optional]

A *Point* object that specifies the position, in pixels, for the position of the timed message dialog on the screen. By default the dialog is centered in the screen. If *p* is used, the point specifies the position of the upper left corner of the dialog.

Example 1:

The following example shows a window with the title of *Infomation* for a duration of 3 seconds:

```
dlg = .TimedMessage~New("Application will be started, please wait", -
                        "Information", 3000)
dlg~execute
drop dlg
```

Example 2:

The following example shows an introductory window that displays while an application is doing its time consuming start up routine. Since this start up process can vary in time depending on the individual system, the window is set to display indefinitely. When the start up process is finished, the programmer dismisses the window and destroys the dialog manually.

```
dlg = .TimedMessage~New("The WidgetCreator Application - loading ...", -
                        "The Widget Factory", -1)
dlg~execute
ret = doStartup()
dlg~ok
drop dlg
if ret <> 0 then do
  ...
end
```

Example 3:

The following example is a variation on **Example 2**. In this case the Widget Factory executives decided that they want their WidgeCreator splash screen to always display for 2 seconds to the customer and then close. The early reply argument is used so that the start up routine executes immediately. After 2 seconds the splash screen dismisses and the dialog is destroyed automatically.

Note

It is not necessary to explicitly drop the dlg variable. That is shown in the other examples to emphasize the point that the dialog has been destroyed.

```
dlg = .TimedMessage~New("The WidgetCreator Application - loading ...", -
                        "The Widget Factory", 2000)
dlg~execute
if doStartup() <> 0 then do
  ...
end
```

30.1.2.10.2. defineDialog

```
>>-aTimedMessage~DefineDialog----------------------------------><
```

The defineDialog method is called by the create method of the parent class, PlainUserDialog, which in turn is called at the very beginning of Execute. You do not have to call it. However, you may want to over-ride it in your subclass to add more dialog controls to the window. If you over-ride it, you have to forward the message to the parent class by using the keyword super.

Example:

The following example shows how to subclass the TimedMessage class and how to add a background bitmap to the dialog window:

```
::class MyTimedMessage subclass TimedMessage inherit DialogExtensions

::method defineDialog
   self~backgroundBitmap("mybackg.bmp", "USEPAL")
   self~DefineDialog:super()
```

30.1.2.10.3. execute

```
>>-aTimedMessage~Execute---------------------------------------><
```

The execute method creates and shows the message window. If the specified *duration* is not negative, this method destroys the dialog automatically when the duration is up. If the duration is less than 0, then the programmer must destroy the dialog manually by invoking the **ok** method. See this *example* for a code snippet showing the procedure.

30.1.3. Public Routines

The routines listed in this section can be used in any Rexx program. They are designed to be even easier to use than the standard dialogs. Simply add a requires directive for the **`ooDialog.cls`** file in a Rexx program to use any of these routines.

```
::requires ooDialog.cls
```

30.1.3.1. AskDialog Routine

Pops up a message box containing the specified text, a Yes button, and a No Button.

```
>>-AskDialog(--question--+-------------------+--)-------------><
                         +--, defaultbutton--+
```

Arguments:

The only argument is:

question

Text to be displayed in the message box.

defaultbutton

Specifies which button, the Yes or the No button, has the default focus when the message box pops up. This argument can be Yes or No and is optional. If the argument is omitted, the Yes button will be given the focus. Only the first letter of the option is needed and case is not significant.

Return Values:

0

The No button has been selected.

1

The Yes button has been selected.

30.1.3.2. CheckList Routine

A shortcut function to invoke the *CheckList* dialog:

```
>>--CheckList(--message--,--title--,--labels--,-+-----------+---->
                                                +-,--checks-+

>---+------------------------+--)------------------------------><
    +-,--+-----+--+--------+-+
         +-len-+  +-,--max-+
```

Arguments:

The arguments are similar to what is described in the *init*) method of the **`CheckList`** class. However, instead of stems, arrays are passed into and returned from the function.

message

A text string that is displayed on top of the check box group. It could be used, for example, to give the user a hint as to what to do.

title
: A text string for the dialog's title

labels
: An array containing the labels, in order, for each of the check boxes. The dialog will contain a check box for each label.

checks
: This argument is an array that allows you to pre-check any of the check boxes. For each index in this array whose item is **.true**, (or 1,) the check box at the corresponding index in the **labels** array will be checked. For any index that is not **.true** , the corresponding check box will not be checked. This means that the programmer only needs to put a **.true** at the indexes where he wants the check boxes checked. All the other indexes can be left empty.

len
: Determines the length (in dialog units) of the check boxes and labels. If omitted, the size is calculated to fit the largest label.

max
: The maximum number of check boxes in one column. If there are more check boxes than max, that is, if labels has more items than the value of max, the check boxes will be put into columns. Each column will contain no more than the number of check boxes specified by max.

Return value:

If the user cancels the dialog, then **.nil** is returned. If the user clicks the okay button, then an array is returned. The array will be the same size as the input **labels** array. Each index of the returned array will contain **.true** if the corresponding check box was checked by the user or will contain **.false** if the check box was not checked.

Example

The following example show how to use the **CheckList** public routine:

```
  weekdays = .array~of("Monday","Tuesday","Wednesday","Thursday", -
                       "Friday","Saturday","Sunday")

  -- Monday and Tuesday will be checked, all the rest will not be checked.
  checks = .array~of(.true, .true, .false)
  checks[5] = 0
  checks[7] = 'a'

  -- The labels will be 60 dialog units wide and there will be at most 4 check
  -- boxes in a column.  (Only 7 check boxes, so 2 columns, the first with 4
  -- check boxes, the second with 3.)

  days = CheckList("Check the days", "Working Days", weekdays, checks, 60, 4)

  if days <> .Nil then do i = 1 to days~items
    if days[i] then say "Working day =" weekdays[i]
  end

::requires "ooDialog.cls"
```

30.1.3.3. ErrorDialog Routine

Pops up a message box containing the specified text, an OK button, and an error symbol.

```
>>-ErrorDialog(--error_text--)--------------------------------><
```

Argument:

The only argument is:

error_text

Text to be displayed in the message box.

30.1.3.4. FileNameDialog Routine

```
>>-FileNameDialog(-+-----+-+----------+-+-------+-+--------+-+---------+-->
                   +-sel-+ +-,-parent-+ +-,-msk-+ +-,-type-+ +-,-title-+

>-----------------+----------+-+---------+-+-----------+--)---------------><
                  +-,-defExt-+ +-,-multi-+ +-,-sepChar-+
```

Causes a file selection dialog box to appear. File selection dialogs are used to allow the user to pick a file or set of files to open, or to pick a file to save to.

Arguments:

The arguments are:

sel [optional]

This argument can be used to pre-select a file and / or have the open or save dialog start in a specified directory.

This argument initializes the open or save dialog by selecting the specified file and placing its name in the *File* edit control of the dialog. In addition this argument can also be used to specify the initial directory (folder) the dialog opens in.

When this argument contains no path information, then the *File* name edit control will contain the argument and the initial directory is determined by the operating system. Which directory the operating system picks varies slightly between versions of Windows. But, in general, it will be a directory previously used in one of the Windows common dialogs, or a directory in the users My Documents folder.

If the argument contains file name and path information, the initial directory will be determined by the path information and the *File* name edit control will contain the file information. When the argument contains only path information and no file name information, i.e. the argument ends with a backslash \, then the initial directory will be that path and the *File* name edit control will be left blank.

In cases where the argument contains incorrect path information, the initial directory is determined as if no path information had been supplied and the entire value of the argument is placed in the File name field.

Note that Windows treats normal wild card characters as valid for file names. Therefore, values such as *C:\*.** or *C:\Windows\*.exe* are acceptable for this argument.

parent [optional]

The window *handle* of a parent window for the open or save file dialog. Normally the programmer would use the handle of a dialog in her program. The operating system disables the parent window until the user closes the open or save dialog.

msk [optional]

This argument consists of pairs of filter strings where each of the filter strings is separated with a **`'0'x`** character and the pairs of strings within a filter are also separated with a **`'0'x`** character. The first string in each pair describes the filter and the second string in the pair is a list of file patterns to use for the filter. For instance, if the file pattern filter is **`*.txt`** then only files with the extension of **`.txt`** are displayed in the dialog. The file pattern can consist of any valid file name characters and and the asterisk (*) wildcard character. But the pattern can not contain any space characters.

An example of how to construct this string is as follows:

```
  delimiter = '0'x

  pair1 = "DLL Files (*.dll)"delimiter"*.dll"delimiter
  pair2 = "Class Files (*.cls)"delimiter"*.cls"delimiter

  filter = pair1 || pair2
```

To specify multiple file extension patterns for a single display string, use a semicolon to separate the patterns, for example:

```
  delimiter = '0'x

  pair1 = "Text Files (*.txt)"delimiter"*.txt;*.bak;*.doc"delimiter
  pair2 = "Class Files (*.cls)"delimiter"*.cls"delimiter

  filter = pair1 || pair2
```

The operating system does not change the order of the filters. In the open or save dialog the filters are displayed in the File Types combo box in the order specified in the *mask* argument.

To have the dialog box not display any filters, use the empty string for this argument. The default filter string when this argument is omitted is:

```
  delimiter = '0'x

  "Text Files (*.txt)"delimiter"*.txt"delimiter"All Files
 (*.*)"delimiter"*.*"delimiter
```

type [optional]

A keyword specifying if the *File Open* or *File Save* dialog is displayed. By default the *File Open* is used. The keyword can be *LOAD* for the file open dialog or *SAVE* for the file save dialog. Only the first letter of the argument is needed and case is not significant.

If the *File Save* dialog is not explicitly requested, the *File Open* dialog is always presented. For historic reasons this argument can also be **`1`** for the *File Open* dialog or **`0`** for the **`File Save`** dialog.

title [optional]

The window title for the save or open file dialog. The default is *Open a File* or *Save File As*, depending on what is specified for the *type* argument.

defExt [optional]

The default file extension. The *File Open* or *File Save* dialog append this extension if the user does not type an extension. Although this extension can be any length, the operating system only uses the first 3 characters. The string must not include the *.* character.

If this argument is omitted, *txt* is used. To prevent the operating system from appending an extension, the programmer should use the empty string for this argument.

multi [optional]

Specifies if the *File Open* dialog allows selection of multiple files. Normally, the open dialog allows the user to select one file only. This can be changed by specifying the keyword *MULTI* for this option. Only the first letter is needed and case is not significant.

This option is ignored for *File Save* dialogs. If the multiple file selection is enabled and the user selects multiple files, the returned result is then: **`path file1 file2 file3 ...`**

sepChar [optional]

Specifies the separation character for the returned path and file names. This is needed for file names with blank characters. If this argument is omitted, the separation character is a blank. If the argument is specified, the returned path and file name string uses this separation character. For example, if you specify # as the separation character, the return string might look as follows:

```
C:\WINNT#file with blank.ext#fileWithNoBlank.TXT
```

Return value:

Returns the selected file name or 0 if the user cancels the dialog. Open file dialogs with multiple selection return **`path file1 file2 file3 ...`** when the user selects multiple files.

Remarks:

If the user changes directory with the *File Open* or **`File Save`** dialog, then the current directory of the Rexx program will be changed. This is the behavior of the underlying Windows dialog, which is designed to work that way. The ooDialog programmer should be aware of this behavior and insert code to change the directory back if that is needed. For example:

```
curdir = directory()
file = FileNameDialog("*.pdf", , "Adobe PDF Files"'0'x"*.pdf"'0'x)
call directory curdir
```

30.1.3.5. InfoDialog Routine

Pops up a message box containing the specified text and an OK button.

```
>>-InfoDialog(--info_text--)----------------------------------><
```

Argument:

The only argument is:

info_text

Text to be displayed in the message box.

30.1.3.6. InputBox Routine

A shortcut function to invoke an *InputBox* dialog as a function:

```
say "Your name:" InputBox("Please enter your name","Personal Data")
```

The parameters are described in the *init* method of the**InputBox** class.

30.1.3.7. IntegerBox Routine

A shortcut to invoke an *IntegerBox* dialog as a function:

```
say "Your age:" IntegerBox("Please enter your age","Personal Data")
```

The parameters are described in the *init* method of the **InputBox** class.

30.1.3.8. ListChoice Routine

The ListChoice function provides a shortcut to invoke a *ListChoice* dialog:

```
day = ListChoice("Select a day","My favorite day", ,
                 .array~of("Monday","Tuesday","Wednesday","Thursday", ,
                          "Friday","Saturday","Sunday") , , ,"Thursday")
say "Your favorite day is" day
```

The parameters are described in the *init* method of the **ListChoice** class. However, instead of an input stem an array is passed into the function.

If the user cancels the dialog **.nil** is returned rather than the empty string.

30.1.3.9. MessageDialog Routine

```
>>--MessageDialog(--msg-+--------+-+-------+-+------+-+--------+-+--------+-)---><
                        +-,-hwnd-+ +-,-cap-+ +-,-pb-+ +-,-icon-+ +-,-misc-+
```

Displays a message box on the screen with the options specified.

Arguments:

The arguments are:

msg [required]

The message text to be displayed.

hwnd [optional]

The window *handle* of the owner window for the message box. The owner window is disabled until the user closes the message box. If this argument is omitted, the ooDialog framework will use the last created, topmost, dialog as the owner window. To explicitly choose that there should be no owner window, specify **0** for this argument.

cap [optional]

The caption, or title, of the message box. The default title when omitted is: *ooDialog Application Message*.

pb [optional]

The *pb* argument specifies which push buttons the message box will have. The default if this argument is omitted is a single Ok button. Otherwise, use exactly one of the following keywords, case is not significant:

ABORTRETRYIGNORE	OKCANCEL	YESNOCANCEL

CANCELTRYCONTINUE	RETRYCANCEL
OK	YESNO

ABORTRETRYIGNORE
: The message box wil have three push buttons: **Abort**, **Retry**, and **Ignore**.

CANCELTRYCONTINUE
: The message box wil have three push buttons: **Cancel**, **Try Again**, and **Continue**.

OK
: The message box wil have one push button: **Ok**. This is the default if the argument is omitted.

OKCANCEL
: The message box wil have two push buttons: **Ok** and **Cancel**.

RETRYCANCEL
: The message box wil have two push buttons: **Retry** and **Cancel**.

YESNO
: The message box wil have two push buttons: **Yes** and **No**.

YESNOCANCEL
: The message box wil have three push buttons: **Yes**, **No**, and **Cancel**.

icon [optional]

The *icon* argument specifies the icon placed in the message box. If this argument is omitted then the message box will not display an icon. Note that several keywords may specify the same icon. To have an icon appear in the message box, use exactly one of the following keywords, case is not significant:

EXCLAMATION	QUESTION	QUERY
WARNING	STOP	NONE
INFORMATION	ERROR	
ASTERISK	HAND	

EXCLAMATION
: The message box will have an exclamation-point icon.

WARNING
: The message box will have an exclamation-point icon.

INFORMATION
: The message box will have an icon consisting of a lowercase letter i in a circle.

ASTERISK
: The message box will have an icon consisting of a lowercase letter i in a circle.

QUESTION
: The message box will have a question-mark icon. Microsoft recommends that this icon not be used any more for a number of reasons. If you want your Rexx programs to have the same look and feel of other Windows programs, then you should not use the question-mark icon.

STOP
: The message box will have a stop-sign icon.

ERROR
: The message box will have a stop-sign icon.

HAND
: The message box will have a stop-sign icon.

QUERY
: The message box will have a question-mark icon, same as the QUESTION keyword. Microsoft recommends not using this icon.

NONE
: Used to allow the programmer to explicitly state that the message box should not have an icon. Omitting the *icon* argument altogether has the same effect.

misc [optional]

The *misc* argument specifies a number of other attributes that can be set for the message box. The argument consists of a blank separated string of keywords from the lists below. The default if this argument is omitted is **`"DEFBUTTON1 APPLMODAL"`**.

The keywords are divided up into four groups. Except for the *other miscellaneous* group, only one keyword from each group should be used, case is not significant. If more than one keyword in a group is used, the behavior is undefined.

These keywords are used to select the default push button when the message box is shown. Use only one keyword from this group.

DEFBUTTON1	DEFBUTTON3
DEFBUTTON2	DEFBUTTON4

DEFBUTTON1
: The first button is the default button. This is the default value if none of the other default button keywords are used.

DEFBUTTON2
: The second button is the default button.

DEFBUTTON3
: The third button is the default button.

DEFBUTTON4
: The fourth button is the default button.

These keywords are used to indicate the *modality* of the message box. **Note** that the description of the modality behaviour here assumes that the window specified by the *hwnd* argument belongs to the programmer's Rexx application. Use only one keyword from this group.

APPLMODAL	SYSTEMMODAL	TASKMODAL

APPLMODAL
: Application Modal: The user must close the message box before continuing to work in the window specified by the *hwnd* argument. However, the user can move to windows on the desktop not part of the application and work in those windows.

  Depending on the hierarchy of windows in the application, the user may be able to move to other windows within the application. However, all child windows of the owner window of the message box are automatically disabled.

APPLMODAL is the default if neither SYSTEMMODAL nor TASKMODAL are specified.

SYSTEMMODAL
System Modal: The modality for this keyword is the same as APPLMODAL except that the message box will have the topmost style. This means that the message box will stay on top of other windows not part of the application, if the user moves to those windows.

Microsoft suggests that this type of message box be used to notify the user of serious, potentially damaging errors that require immediate attention. SYSTEMMODAL has no effect on the user's ability to interact with windows other than those associated with the *hwnd* argument.

TASKMODAL
Task Modal: This is the same as APPLMODAL except that all the top-level windows belonging to the current thread are disabled if the *hwnd* argument is 0. Use this keyword when the programmer does not have a window handle available but still needs to prevent input to other windows in the application.

Other miscellaneous keywords. One or more keywords from this group can be used..

DEFAULTDESKTOP	RTLREADING	TOPMOST
RIGHT	SETFOREGROUND	SERVICENOTIFICATION

DEFAULTDESKTOP
Same as the desktop of the interactive window station. The programmer should consult the Windows *documentation* when considering to use this keyword.

RIGHT
The message text is right-justified.

RTLREADING
Displays the message and caption text using right-to-left reading order on Hebrew and Arabic systems.

SETFOREGROUND
The operating system, internally, sets the message box as the foreground window.

TOPMOST
The message box will have the topmost window style. This will keep the message box above all other windows on the desktop.

SERVICENOTIFICATION
The application dialog calling the *MessageDialog* routine is a service notifying the user of an event. The *MessageDialog* displays a message box on the current active desktop, even if there is no user logged on to the computer.

ooDialog and Rexx programs in general are not particularly suited to implement Windows services. So it is unlikely that this keyword will be used.

HELP
Adds a **Help** push button to the message box, in addition to the push buttons specifed in the *pb* argument. When the user clicks the **Help** button or presses F1, the operating system sends a HELP *event* notification to the window specified by the *hwnd* argument. If this window is an ooDialog dialog, the programmer can capture this notification using the *connectHelp* method.

Return value:

MessageDialog returns the Microsoft defined resource ID of the button pushed by the user, and 0 on error. The programmer can use a *constant* of the *dialog* object to determine the return. The return values are:

0
: An error occurred. the *.systemErrorCode* will be set.

1 IDOK
: The **Ok** button was pushed.

2 IDCANCEL
: The **Cancel** button was pushed.

3 IDABORT
: The **Abort** button was pushed.

4 IDRETRY
: The **Retry** button was pushed.

5 IDIGNORE
: The **Ignore** button was pushed.

6 IDYES
: The **Yes** button was pushed.

7 IDNO
: The **No** button was pushed.

8 IDCLOSE
: The **Close** button was pushed.

9 IDHELP
: The **Help** button was pushed.

10 IDTRYAGAIN
: The **Try Again** button was pushed.

11 IDCONTINUE
: The **Continue** button was pushed.

Remarks:

When the message box has a **Cancel** button, IDCANCEL is also returned if the user uses the ESC key. If the message box does not have a **Cancel** button, then the user can not close the message box by using the ESC key.

The *AskDialog*, *InfoDialog*, and *ErrorDialog* routines are all implemented using the **`MessageDialog`** routine. While they are simplier to use, they are not very flexible. In addition because of the way *modal* dialogs are implemented, it is possible that those routines will not disable the correct dialog. If the programmer encounters problems with the wrong dialog being disabled, the solution is to use the **`MessageDialog`** routine and specify the proper window handle of the owner window.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example is from a Wizard 97 program that warns the user when she is about to change the page and has not completed all of the steps on the page she is on.

```
::method wizNext unguarded
  expose selectedMovies movieTheaters movieCombo
  use arg propSheet

  do movie over selectedMovies
    if movieTheaters[movie] == .nil then do

      msg = "Warning.  You have not selected a movie" || .endOfLine ||           -
            "theater for all selected movies."        || .endOfLine~copies(2) || -
            "Are you sure you want to go to"          || .endOfLine ||           -
            "the next page?"

      title = "Theater Selection is not Complete"
      buttons = "YESNO"
      icon = "WARNING"
      style = "DEFBUTTON2 TASKMODAL"

      ans = MessageDialog(msg, propSheet~dlgHandle, title, buttons, icon, style)

      if ans == .PlainBaseDialog~IDYES then return 0

      movieCombo~select(movie)
      self~setRadioButtons(movie)
      return -1
    end
  end

  return 0
```

30.1.3.10. MultiInputBox Routine

Provides a shortcut function to invoke a *MultiInputBox* dialog:

```
res = MultiInputBox("Enter your address","Personal Data", ,
                    .array~of("&First name","Last &name","&City"), ,
                    .array~of("John","Smith","San Jose"), 100)
if res \= .Nil then do entry over res
                       say "Address-line[]= " entry
                    end
```

The parameters are described in the *init* method of the **`MultiInputBox`** class. However, instead of using stems, arrays are used for input.

The user's input to the dialog is also returned in an array. Note this other difference from the MultiInputBox dialog: if the user cancels the dialog **`.nil`** is returned rather than the empty string.

30.1.3.11. MultiListChoice Routine

Provides a shortcut to invoke a *MultiListChoice* dialog as a function:

```
days = MultiListChoice("Select days","My TV Days", ,
                       .array~of("Monday","Tuesday","Wednesday", ,
                              "Thursday","Friday","Saturday","Sunday"), , ,"2 5")
if days <> .Nil then do day over days
```

```
                      say "TV day =" day
                   end
```

The parameters are described in the *init* method of the **`MultiListChoice`** class. However, instead of stems, arrays are passed into the function.

If the user cancels the dialog, **`.nil`** is returned rather than the empty string.

30.1.3.12. PasswordBox Routine

A shortcut to invoke a *PasswordBox* dialog as a function:

```
pwd = PasswordBox("Please enter your password","Security")
```

The parameters are described in the *init* method of the **`InputBox`** class.

30.1.3.13. SingleSelection Routine

```
>>-SingleSelection(--msg-,-title-,-labels-+----------+-+-----------+-+-------+--)--><
                                          +-,-preSel-+ +-,-rbWidth-+ +-,-max-+
```

The SingleSelection function provides a shortcut to invoke a *SingleSelection* dialog. However, the *labels* argument is an array, rather than a stem as it is in the **`SingleSelection`** class.

Arguments:

The arguments are:

msg [required]

A text string that is displayed above the radio button group. Typically, the text would be used to give the user the purpose of the radio buttons, or to prompt the user as to what to do.

title [required]

A title, the caption bar text, for the dialog that will be invoked.

labels [required]

An array with indexes one through the number of radio buttons desired. The value for each index is the label for the corresponding radio button. For instance the value at index 2 of the array would be set to the label for the second radio button.

preSel [optional]

This argument can be used to preselect, (that is to check,) one radio button. A value of 1 checks the first radio button, a value of 2 checks the second, and so on. If this argument is omitted, the first radio button is checked.

rbWidth [optiona]

Specifies the width, in dialog units, of the radio button controls. If omitted, the width is calculated so that the longest label will not be clipped. If used, all radio buttons are set to this width without regard as to whether the label will be clipped or not.

max [optional]

The maximum number of radio buttons in one column. If there are more radio buttons than *max*, that is, if the *labels* argument has more indexes than the value of *max*, the radio buttons are positioned in columns. Each column will have *max* radio buttons and there will be as many columns as needed to display all the radio buttons.

Return value:

If the user cancels the dialog, the empty string is returned. If the user closes the dialog with *Ok* the return is the index in the *labels* array of the selected radio button. 1 for the first radio button, 2 for the second radio button, 8 for the eighth radion button, etc., etc..

Remarks:

When the *SingleSelection* routine is called, the ooDialog framework, internally, does a number of calculations to determine the optimal size of the dialog controls and dialog. The framework then positions all the dialog controls such that none of the radio button labels, message, and title of the dialog are clipped.

The one exception to this is if the programmer specifies the width of the radio button controls using the optional *rbWidth* argument. In this case the width of the radio buttons is always set to what the programmer has specified. It is possible that the radio buttons will be clipped, but that is now the responsibility of the programmer.

Example:

The following example shows a dialog that contains a two-column radio button group. The sixth radio button (the button with the label Jun) is checked, (preselected.)

```
months = .array~of("Jan","Feb","Mar","Apr","May","Jun",  -
                   "Jul","Aug","Sep","Oct","Nov","Dec")
dlgTitle = "The Month You Were Born"
prompt = "Select the month of your birth:"
selectedRadioButton = 3

m = SingleSelection(prompt, dlgTitle, months, selectedRadioButton, , 6)
if m == "" then
    say "The user canceled the dialog."
else
    say "Born in month" m "=" months[m]

::requires "ooDialog.cls"
```

30.1.3.14. TimedMessage Routine

The TimedMessage routine provides a shortcut to invoke a *TimedMessage* dialog as a function:

```
ret = TimedMessage("We are starting...","Please wait",3000)
```

The parameters are the same as described in the *init* method of the **`TimedMessage`** class.

The function returns 0, always, when the duration is non-negative. When the duration is less than 0, the function returns a reference to the **`TimedMessage`** dialog object. This is needed in order for the programmer to manually dismiss the dialog. Below are the same 3 *examples* listed for the **`TimedMessage`** dialog, modified to work with the public routine:

Example 1:

The following example shows a window with the title of *Infomation* for a duration of 3 seconds:

```
ret = timedMessage("Application will be started, please wait", -
                   "Information", 3000)
```

Example 2:

The following example shows an introductory window that displays while an application is doing its time consuming start up routine. Since this start up process can vary in time depending on the individual system, the window is set to display indefinitely. When the start up process is finished, the programmer dismisses the window and destroys the dialog manually. Note how a reference to the **`TimedMessage`** dialog is returned. The programmer only needs to use this to dismiss / destroy the dialog

```
dlg = timedMessage("The WidgetCreator Application - loading ...", -
                   "The Widget Factory", -1)
ret = doStartup()
dlg~ok
if ret <> 0 then do
  ...
end
```

Example 3:

The following example is a variation on **Example 2**. In this case the Widget Factory executives decided that they want their WidgeCreator splash screen to always display for 2 seconds to the customer and then close. The early reply argument is used so that the start up routine executes immediately. After 2 seconds the splash screen dismisses and the dialog is destroyed automatically.

```

ret = timedMessage("The WidgetCreator Application - loading ...", -
                   "The Widget Factory", 2000)

if doStartup() <> 0 then do
  ...
end
```

30.2. Mouse Class

In Windows the mouse is one of the important user input devices for programs. A program receives mouse input in the form of *message*s that are sent or posted to the windows of the program. In addition the operating system provides a number of functions to work with the mouse. These functions allow the programmer to change the mouse behavior, configure the mouse, change its appearence, etc..

Mouse Cursor: As the user moves the mouse across the screen, the operating system paints a bitmap on the screen that gives the user feedback as to the position of the mouse. This bitmap is called the *mouse cursor*. The cursor has a single pixel point in the bitmap called the *hot spot*. The position of the hot spot is considered to be the position of the mouse. All mouse messages are, normally, sent to the window beneath the hot spot, whether or not that window is active or has the keyboard focus.

In the ooDialog framework, the **`Mouse`** class provides all the methods to work with the mouse and the mouse cursor. Because all mouse messages are sent by the operating system to a single window, in ooDialog each instantiated mouse object is affiliated with a specific window object. I.e., a specific dialog or dialog control object.

30.2.1. Hit Test Keywords

Windows has defined a number of numeric codes that are used to determine where the mouse cursor is positioned. These codes are used for what is commonly called a *hit test*. ooDialog defines a keyword for each of these codes.

Table 30.7. Possible *Hit Test* Values

Keyword	Description
ERROR	On the screen background or on a dividing line between windows (same as NOWHERE.)
TRANSPARENT	In a window currently covered by another window in the same thread.
NOWHERE	On the screen background or on a dividing line between windows.
CLIENT	In a client area.
CAPTION	In a title bar.
SYSMENU	In a window menu or in a Close button in a child window.
GROWBOX	In a size box.
MENU	In a menu.
HSCROLL	In a horizontal scroll bar.
VSCROLL	In the vertical scroll bar.
MINBUTTON	In a Minimize button.
MAXBUTTON	In a Maximize button.
LEFT	In the left border of a resizable window (the user can click the mouse to resize the window horizontally).
RIGHT	In the right border of a resizable window (the user can click the mouse to resize the window horizontally).
TOP	In the upper-horizontal border of a window.
TOPLEFT	In the upper-left corner of a window border.
TOPRIGHT	In the upper-right corner of a window border.
BOTTOM	In the lower-horizontal border of a resizable window (the user can click the mouse to resize the window vertically).
BOTTOMLEFT	In the lower-left corner of a border of a resizable window (the user can click the mouse to resize the window diagonally).
BOTTOMRIGHT	In the lower-right corner of a border of a resizable window (the user can click the mouse to resize the window diagonally).
BORDER	In the border of a window that does not have a sizing border.
OBJECT	Microsoft does not describe what this means.
CLOSE	In a Close button.
HELP	In a Help button.

30.2.2. Method Table

The following table lists the class and instance methods of the **`Mouse`** class:

Table 30.8. Mouse Class Method Reference

Method	Description
Constant Methods	The **`Mouse`** class provides a single *constant* value through the **`::constant`**directive.

Method	Description
Class Methods	**Class Methods**
new	Instantiates a new mouse object.
doubleClickTime	Retrieves the current double-click time for the mouse.
loadCursor	Returns one of the shared pre-defined system cursors.
loadCursorFromFile	Creates a cursor based on data from a file.
setDoubleClickTime	Sets the double-click time for the mouse.
swapButton	Reverses or restores the meaning of the left and right mouse buttons.
Instance Methods	**Instance Methods**
appStarting	Sets the mouse cursor to the system's default *application starting* cursor, which is usually the standard arrow with a small hourglass.
arrow	Sets the mouse cursor to the system's *default* cursor, which is usually the arrow cursor.
capture	Sets the mouse capture to the window of this mouse instance.
clipCursor	Restricts the cursor position to a rectangular area on the screen.
connectEvent	Connects mouse related *event* notifications to a method in the Rexx dialog.
cross	Sets the mouse cursor to the system's default *high precision* cursor, which is usually the crosshair cursor.
dragDetect	Used to determine if the user has started a drag operation.
getCapture	Retrieves the window handle, if any, that currently has captured the mouse.
getClipCursor	Retrieves the screen coordinates of the rectangular area to which the cursor is confined.
getCursorPos	Retrieves the current cursor position in pixels as a **`Point`** object.
isButtonDown	Determines if a mouse button is pressed.
no	Sets the mouse cursor to the system's default *not allowed* cursor, which is usually a circle with a slash across it.
releaseClipCursor	Releases the cursor so that it is free to move anywhere on the screen.
releaseCapture	Releases the mouse capture from the window which had previously captured the mouse.
restoreCursor	Restores the cursor for the window of this mouse instance.
setCursor	Sets the cursor for the window of this mouse instance.
setCursorPos	Moves the mouse cursor to the specified position.
showCursor	Displays or hides the cursor.
trackEvent	Initiates, or queries, mouse tracking behavior.
wait	Sets the mouse cursor to the system's default *busy* cursor, which is usually the hourglass cursor.

30.2.3. Constant Methods

The **`Mouse`** class provides a single *constant* value through the use of the **`::constant`** directive. The constant is listed in the following table.

Table 30.9. Mouse Class Constant Reference

Constant Symbol	Description
HOVER_DEFAULT	The default hover time out. Can be used for the *hoverTime* argument in the *trackEvent* method.

30.2.4. new (Class Method)

```
>>--new(--windowObj--)---------------------------><
```

Instantiates a new **Mouse** object. Each mouse object is associated with a single *underlying* window, the window of the Rexx object used to instantiate the mouse object.

Arguments:

The single argument is:

windowObj [required]

A Rexx object that has an underlying Windows window. In all practical terms this is either a Rexx *dialog* object or a Rexx dialog *control* object.

Return value:

The return is a new **Mouse** object.

Remarks:

The operating system sends all mouse related *message*s to the window the mouse is over. Because of this, each Rexx mouse object is associated with a specific window. This is important for the programmer to understand, especially when connecting mouse event notifications. Since all dialog controls are windows, when the mouse is over a dialog, most messages go to one of the dialog controls that make up the dialog. Therefore, if the programmer is interested is the mouse event notifications, much of the time the programmer will need to use a mouse object associated with a dialog control, not a mouse associated with the dialog.

All mouse event notifications are connected through the *connectEvent* method of the **Mouse** class, not through one of the connect event methods of the dialog.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example constructs a mouse object using the dialog object and then connects the double click mouse event to a method in the dialog. When the user double clicks the primary mouse button on the background of the dialog, not on a dialog control, the event handler is invoked:

```
::method init
  ...

  mouse = .Mouse~new(self)
  mouse~connectEvent('LBUTTONDBLCLK', onDoubleClick)
  ...

::method onDoubleClick unguarded
  use arg state, position, mouse

  -- Do something ...
```

30.2.5. doubleClickTime (Class Method)

```
>>--doubleClickTime------------------------------><
```

Retrieves the current double-click time for the mouse.

Arguments:

There are no arguments for this method.

Return value:

The return is the current double-click time in milliseconds.

Remarks:

A double-click is a series of two clicks of the mouse button, the second occurring within a specified time after the first. The double-click time is the maximum number of milliseconds that may occur between the first and second click of a double-click. The double-click time is a system-wide parameter.

Example:

This simple example prints the double-click time to the screen:

```
 ::method showDoubleClkTime

   say 'The system-wide mouse double-click time is:' .Mouse~doubleClickTime
'milliseconds.'

 /* Output might be:

 The system-wide mouse double-click time is: 340 milliseconds.

 /*
```

30.2.6. loadCursor (Class Method)

```
>>--loadCursor(--cursorName--)-------------------><
```

Returns one of the shared pre-defined system cursors.

Arguments:

The single argument is:

cursorName [required]

Exactly one of the following keywords to indicate which system cursor is desired, case is not significant:

APPSTARTING	IBEAM	SIZENWSE
ARROW	NO	SIZEWE
CROSS	SIZEALL	UPARROW
HAND	SIZENESW	WAIT
HELP	SIZENS	

APPSTARTING

Standard arrow and small hourglass.

ARROW	Standard arrow.
CROSS	Crosshair.
HAND	Hand.
HELP	Arrow and question mark.
IBEAM	I-beam.
NO	Slashed circle.
SIZEALL	Four-pointed arrow pointing north, south, east, and west.
SIZENESW	Double-pointed arrow pointing northeast and southwest.
SIZENS	Double-pointed arrow pointing north and south.
SIZENWSE	Double-pointed arrow pointing northwest and southeast.
SIZEWE	Double-pointed arrow pointing west and east.
UPARROW	Vertical arrow.
WAIT	Hourglass.

Return value:

The requested cursor as a *Image* object on success, or 0 on error.

Remarks:

The returned **`Image`** object is a shared image and should not be released. The ooDialog framework will ignore any requests to release the object.

Note that the description for the system cursors is the description for the default pre-defined cursors. The user has the ability to change any of the default cursors to a custom cursor.

The loaded cursor is suitable to use in the *setCursor* method. Note that the system cursors can also be loaded directly using the *Image* class's *getImage* method.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

Example:

This example loads the *No* system cursor and checks for error. In reality, there is almost no chance of failure provided the correct keyword is used.

```
::method init
  ...

  noDropCursor = .Mouse~loadCursor("NO")
  if noDropCursor == 0 then do
    msg = 'Error loading No Drop Cursor, system error code: ' || -
          .systemErrorCode SysGetErrortext(.systemErrorCode)

    title = 'Drag N Drop Rexx - Unrecoverable Error'

    ret = MessageDialog(msg, , title, 'OK', "EXCLAMATION")

    self~initCode = 1
    return self~initCode
  end
```

30.2.7. loadCursorFromFile (Class Method)

```
>>--loadCursorFromFile(--fileName--)-------------><
```

Creates a cursor based on data from a file.

Arguments:

The single argument is:

fileName [required]

The name of the file containing the cursor data. This argument can either be a fully qualified or a relative file name.

Return value:

The requested cursor as a *Image* object on success, or 0 on error.

Remarks:

Note that the source file for the cursor must actually be a cursor file or the OS will refuse to load it. E.g., the file can not be simply any image file, trying to load a bitmap, or even an icon file, as a cursor will fail.

The returned *Image* object is not a shared image and can be released when no longer needed, if desired to free up some (small) amount of system resources.

The loaded cursor is suitable to use in the *setCursor* method. Note that the system cursors can also be loaded directly using the *Image* class's *getImage* method.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

Example:

This example is similar to the *loadCursor*, but it loads a custom cursor from a file:

```
::method init
  ...
  dropOkCursor = .Mouse~loadCursorFromFile("dragging.cur")
  if dropOkCursor == 0 then do
    msg = 'Error loading Drop Ok Cursor, system error code: ' || -
          .systemErrorCode SysGetErrortext(.systemErrorCode)

    title = 'Drag N Drop Rexx - Unrecoverable Error'

    ret = MessageDialog(msg, , title, 'OK', "EXCLAMATION")

    self~initCode = 1
    return self~initCode
  end
```

30.2.8. setDoubleClickTime (Class Method)

```
>>--setDoubleClickTime(--+------------+--)-------><
                         +-,-interval-+
```

Sets the double-click time for the mouse.

Arguments:

The single argument is:

interval [optional]

The time in milliseconds to set the double-click time to. If this is 0, the default system double-click time is restored, which is 500 milliseconds. If omitted the default for *interval* is 0.

Return value:

True on success, false on error.

Remarks:

A double-click is a series of two clicks of the mouse button, the second occurring within a specified time after the first. The double-click time is the maximum number of milliseconds that may occur between the first and second click of a double-click. The double-click time is a system-wide parameter.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

Example:

This example checks the current double-click time and if it is below 500 milliseconds, it resets the time to the usual system default.

```
::method resetDefaults private

  ...

  if .Mouse~doubleClickTime > 500 then .Mouse~setDoubleClickTime

  ...
```

30.2.9. swapButton (Class Method)

```
>>--swapButton(--+--------+--)-------------------><
                 +--swap--+
```

Reverses or restores the meaning of the left and right mouse buttons.

Arguments:

The single argument is:

swap [optional]

Specifies whether the meaning of the mouse buttons are reversed or restored.

If true, the left button generates right-button messages and the right button generates left-button messages. If false, the buttons are restored to their original meanings. The default is true.

Return value:

If the buttons were swapped before this method is invoked, the return is true, if the buttons were not swapped, the return is false.

Remarks:

Button swapping is provided by Windows as a convenience to people who use the mouse with their left hands. The button swapping functionality is usually done through the Control Panel only. Although any application is free to use this functionality, the mouse is a shared resource and reversing the meaning of its buttons affects the entire system.

Note that the way the operating system handles button swapping makes this mostly transparent to the programmer. When the programmer is interested in receiving a mouse event notification, say the left mouse button clicked event, then he just needs to connect the LBUTTONDOWN event and not worry about whether the user has swapped buttons or not. If the user has swapped the mouse buttons, the operating system will generate the LBUTTONDOWN event when the user presses the right hand button.

Arguably, it might be better to call the left physical mouse button the *primary* button or *button1*, Windows does not do this and this documentation uses the Windows naming conventions.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode* to 0, but there are no system errors that would change the value from 0.

Example:

This example comes from a program my nephew wrote and set to automatically run at 12:01 am, April 1st on his brother's computer. He thought it was hilarious, his brother was not amused.

```
/* aprilFools.rex */

  .Mouse~swapButton

  return 0

::requires 'ooDialog.cls'
```

30.2.10. appStarting

```
>>--appStarting----------------------------------><
```

Sets the cursor for the window of this mouse instance to the operating system's predefined *application starting* cursor.

Arguments:

This method takes no arguments.

Return value:

Returns the previous cursor on success, or 0 on error. An error is unlikely.

Remarks:

The *appStarting* method is convenience method. It is exactly equivalent to using the *setCursor* method with the APPSTARTING keyword.

Details

Sets the *.systemErrorCode*.

Example:

This example sets the cursor to the application starting cursor for 2 seconds, then restores it. This could be done, for instance, while the application is displaying a splash screen when it starts up. (A practice that used to be very common in Windows, but you don't see that often anymore.)

```
::method initDialog
  self~startingUp

::method startingUp private unguarded

  reply 0

  mouse = .Mouse~new(self)

  -- This sets the mouse position to a point we know is
  -- not over a dialog control.  If the mouse is over a
  -- dialog control, changing the cursor will have no
  -- effect, unless the user moves it over the dialog
  -- background during the 2 seconds.
  p = .Point~new(30, 30)
  self~client2screen(p)

  mouse~setCursorPos(p)

  oldCursor = mouse~appStarting

  -- Jiggle the mouse so that it repaints immediately
  p = mouse~getCursorPos
  p~incr
  mouse~setCursorPos(p)

  j = SysSleep(2)

  mouse~setCursor(oldCursor)

  -- Jiggle ...
  p = mouse~getCursorPos
  p~decr
  mouse~setCursorPos(p)
```

30.2.11. arrow

```
>>--arrow----------------------------------------><
```

Sets the cursor for the window of this mouse instance to the operating system's predefined *normal* cursor, which is usually the standard arrow.

Arguments:

This method takes no arguments.

Return value:

Returns the previous cursor on success, or 0 on error.

Remarks:

The *arrow* method is convenience method. It is exactly equivalent to using the *setCursor* method with the ARROW keyword.

Details

Sets the *.systemErrorCode*.

Example:

This example just shows syntax, the method is not difficult to understand:

```
mouse = .Mouse~new(self)
oldCursor = mouse~arrow
```

30.2.12. capture

```
>>--capture--------------------------------------><
```

Sets the mouse capture to this mouse's window. Once a window has captured the mouse, all mouse input is directed to the window even when the mouse is moved outside the boundaries of the window.

Arguments:

The method takes no arguments.

Return value:

The handle to the window that previously captured the mouse, or 0 if the mouse was not captured before.

Remarks:

The mouse capture functions work with windows of the same thread. For ooDialog, this means the dialog window and the dialog control windows that belong to the same dialog as this mouse's window. The *capture* method can capture mouse input either when the mouse is over the dialog this mouse's window belongs to, or when the mouse button was pressed while the mouse was over that dialog, and the button is still down. Only one window at a time can capture the mouse.

If the mouse cursor is over a window created by another thread, the system will direct mouse input to the specified window only if a mouse button is down.

Only the foreground window, or a window belonging to the foreground window, can capture the mouse. Also, even if the foreground window has captured the mouse, the user can still click another window, bringing it to the foreground.

Details

Raises syntax errors if incorrect usage is detected.

Sets the *.systemErrorCode* to 0, but there are no operating system errors that would change the variable.

Example:

This example shows a method that initiates a drag and drop operation. It captures the mouse so that all mouse messages are directed to the same window and the program can monitor both the current position of the mouse, and when the mouse button is released:

```
::method doDrag private
  expose oldCursor noDropCursor cursorIsNoDrop dragging
  use arg listView, index, p, mouse

  mouse~capture
  oldCursor = mouse~setCursor(noDropCursor)

  dragItem = .DragItem~new(listView, index, p, mouse)

  cursorIsNoDrop = .true
  dragging       = .true

  return dragItem
```

30.2.13. clipCursor

```
>>--clipCursor(--rect--)-------------------------><
```

The *clipCursor* function restricts the cursor position to a rectangular area on the screen. When, or if, a subsequent cursor position (set by the *setCursorPos* method or a mouse movement) would be outside the rectangle, the operating system automatically adjusts the position to keep the cursor inside the rectangular area

Arguments:

The single argument is:

rect

A *bounding* rectangle specified as a *Rect* object that defines the area the cursor is confined to.

Return value:

Returns true on success, false on error.

Remarks:

The cursor is a shared resource. If an application confines the cursor, it must release the cursor at some point by using *releaseClipCursor* before the user can move to another application using the mouse.

However, although the MSDN documentation does not explicity state this, experimentation shows that if the user brings up the Alt-Tab switching dialog and moves to another application in that manner, the cursor is no longer confined. If the user closes the dialog, the mouse is no longer confined.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

Example:

This example confines the cursor to the client area of the dialog:

```
r = self~clientRect
self~client2screen(r)
mouse~clipCursor(r)
```

30.2.14. connectEvent

```
>>--connectEvent(--event--+------------+--+------------+--+--------+--)-------><
                          +-,-methName-+  +-,-willReply-+  +-,-opts-+
```

Connects mouse related Windows *event* notifications to a method in the Rexx dialog.

Arguments:

The arguments are:

event [required]

Specifies which event to connect. Use exactly one of the following keywords, case is not significant:

CAPTURECHANGED	MBUTTONUP	MOUSEWHEEL
LBUTTONDBLCLK	MBUTTONDBLCLK	RBUTTONDBLCLK
LBUTTONDOWN	MOUSEHOVER	RBUTTONDOWN
LBUTTONUP	MOUSELEAVE	RBUTTONUP
MBUTTONDOWN	MOUSEMOVE	

CAPTURECHANGED

This event notification is sent to the window that is losing the mouse capture. The *CAPTURECHANGED* Event Handler contains information on coding the event handler.

LBUTTONDBLCLK

The event notification is sent when the user double-clicks the left mouse button while the cursor is in the client area of the window. The *General* Mouse Event Handler contains information on coding the event handler.

LBUTTONDOWN

The event notification is sent when the user presses the left mouse button while the cursor is in the client area of the window. The *General* Mouse Event Handler contains information on coding the event handler.

LBUTTONUP

The event notification is sent when the user releases the left mouse button while the cursor is in the client area of the window. The *General* Mouse Event Handler contains information on coding the event handler.

MBUTTONDBLCLK

The event notification is sent when the user double-clicks the middle mouse button while the cursor is in the client area of the window. The *General* Mouse Event Handler contains information on coding the event handler.

MBUTTONDOWN
: The event notification is sent when the user presses the middle mouse button while the cursor is in the client area of the window. The *General* Mouse Event Handler contains information on coding the event handler.

MBUTTONUP
: The event notification is sent when the user releases the middle mouse button while the cursor is in the client area of the window. The *General* Mouse Event Handler contains information on coding the event handler.

MOUSEHOVER
: The event notification is sent when the cursor hovers over the client area of the window for the period of time specified in a prior call to the *trackEvent* method. The *General* Mouse Event Handler contains information on coding the event handler.

MOUSELEAVE
: The event notification is sent when the cursor leaves the client area of the window specified in a prior call to the *trackEvent* method. The *MOUSELEAVE* Event Handler contains information on coding the event handler.

MOUSEMOVE
: The event notification is sent when the cursor moves. The *General* Mouse Event Handler contains information on coding the event handler.

MOUSEWHEEL
: The event notification is sent when the mouse wheel is rotated. The *MOUSEWHEEL* Event Handler contains information on coding the event handler.

NCMOUSEHOVER
: The event notification is sent when the cursor hovers over the non-*client area* of the window for the period of time specified in a prior call to the *trackEvent* method. The *NCMOUSEHOVER* Mouse Event Handler contains information on coding the event handler.

NCMOUSELEAVE
: The event notification is sent when the cursor leaves the non-*client area* area of the window specified in a prior call to the *trackEvent* method. The *MOUSELEAVE* Event Handler contains information on coding the event handler.

RBUTTONDBLCLK
: The event notification is sent when the user double-clicks the right mouse button while the cursor is in the client area of the window. The *General* Mouse Event Handler contains information on coding the event handler.

RBUTTONDOWN
: The event notification is sent when the user presses the right mouse button while the cursor is in the client area of the window. The *General* Mouse Event Handler contains information on coding the event handler.

RBUTTONUP
: The event notification is sent when the user releases the right mouse button while the cursor is in the client area of the window. The *General* Mouse Event Handler contains information on coding the event handler.

methName [optional]

The name of the method in the Rexx dialog that is to be invoked when the event notification is generated. If this argument is omitted, the method name is constructed automatically by the ooDialog framework. The method name is constructed by prepending the word *on* to the event keyword. For example, if the event is LBUTTONDOWN, the constructed method name would be *onLButtonDown*. If the event is MOUSEHOVER, the constructed method name would be *onMouseHover*, etc..

willReply [optional]

The *willReply* argument controls whether the interpreter *waits*) for the reply from the event handler. The default is `.true`, the interpreter waits for the reply. If *willReply* is `.false`, the interpreter does not wait for the event handling method to return a value. See the Remarks section.

opts [optional]

Exactly one of the following keywords, case is not significant. The argument adds some control as to how the ooDialog framework replies to the operating system for the specified event notification. This argument is only applicable in certain cases. See the Remarks section for further details.

REPLYFALSE	SENDTODLG	SENDTOCONTROL

REPLYFALSE

When the mouse window is a dialog window and *willReply* is false, this keyword will cause the reply to the operating system to indicate that the mouse event was not handled. By default the reply to the operating system will indicate the event was handled.

SENDTODLG

When the mouse window is a dialog control window, this keyword indicates that the mouse event notification message should be sent straight to the dialog. In this case, the *methodName* and *willReply* arguments have no effect. No method will be invoked. The event will be handled by the dialog window, if connected.

SENDTOCONTROL

When the mouse window is a dialog control window and *willReply* is false, this keyword will cause the event notification to be passed on to the dialog control. This allows the dialog control to handle the event as it normally would. By default, the reply to the operating system is that the event was handled, and the dialog control never sees the event.

Return value:

True on success, false on error. An error is not likely.

Remarks:

See the sections on *connecting* and *coding* event handlers for additional information on event handlers. Note that it is not possible to instantiate a mouse object tied to a dialog control window until after the *underlying* dialog has been created. Therefore, for dialog control windows the *connectEvent* method is normally invoked in the *initDialog* method.

Note that to properly handle and process mouse event messages, it is usually necessary for the interpreter to wait for the reply from the event handler. If the programmer specifies false for the *willReply* argument, the event handler is likely to behave unpredictably. This is particularly true for the MOUSEMOVE event.

For mouse events, there are several possible ways for the ooDialog framework to reply to the operating system in response to the underlying notification message. The possible replies are dependent on which type of window the mouse event is for, a dialog window or a dialog control window.

dialog window:

When the mouse event is for a dialog window, ooDialog can reply to the operating system in these ways:

1. Tell the operating system the event was processed.
2. Tell the operating system the event was not processed.

dialog control window:

When the mouse event is for a dialog control window, ooDialog can reply to the operating system in these ways:

1. Tell the operating system the event was processed.
2. Have the operating system pass the message directly to the dialog.
3. Have the operating system pass the message to the dialog control it was originally intended for.

The programmer specifies how ooDialog replies to the operating system through a combination of the *willReply* argument, the *opts* argument, and the return value of the event handler. The combinations are summarized in the following table:

Table 30.10. Possible Replies to the Operating System

Event	Will	Reply	opts	Action
Window	Reply	Value	Argument	Taken
dialog	true	**`.true`**	omitted	OS told event processed.
dialog	true	**`.false`**	omitted	OS told event not processed.
dialog	false	**`n/a`**	omitted	OS told event processed.
dialog	false	**`n/a`**	REPLYFALSE	OS told event not processed.
dialog control	true	**`.true`**	omitted	OS told event processed.
dialog control	true	**`.false`**	omitted	Message passed to the dialog control.
dialog control	n/a	**`n/a`**	SENDTODLG	Message sent directly to the dialog.

Event Window	Will Reply	Reply Value	opts Argument	Action Taken
dialog control	false	**n/a**	omitted	OS told event processed.
dialog control	false	**n/a**	SENDTOCONTROL	Message passed to the dialog control.

Details

Raises syntax errors when incorrect usage is detected.

Resets the *.systemErrorCode* to zero. There are no operating system errors that would change the variable.

Example:

This example connects the MOUSEWHEEL event for an edit control, using the SENDTODLG keyword for the *opts* argument. Note that no MOUSEWHEEL event is connected to the dialog. This has the effect of *swallowing* the event. The event notification is intercepted before it reaches the edit control, and no method in the dialog is invoked. When the user scrolls the mouse wheel while over the edit control, nothing happens.

```
::method initDialog

  edit = self~newEdit(IDC_EDIT_FILE)

  mouse = .Mouse~new(edit)
  mouse~connectEvent(MOUSEWHEEL, , , SENDTODLG)

  ...
```

30.2.14.1. CaptureChanged Event Handler

The event handler for the capture changed event is invoked when the window assigned to the mouse object is losing the capture. The event is received even if it is the mouse object itself that is *releaseCapture* the capture. Microsoft's documentation states that the event handler should not attempt to *capture* the mouse in response to this event.

The programmer specifies if the interpreter waits or not for the return from the event handler through the arguments used in the *connectEvent* method.

```
::method onCaptureChanged unguarded
  use arg hwnd, mouse

  return .true
```

Arguments:

The event handling method receives 2 arguments:

hwnd

The window *handle* of the window gaining the mouse capture. This may be 0 if there is no window gaining the capture.

mouse

The Rexx *Mouse* object for the window that received the MOUSEWHEEL event notification.

Return:

In general, return **`.true`** to indicate the event has been processed and **`.false`** to indicate it was not processed. However, the correct response is dependent on how the event was connected, and is summarized in the *Remarks* section for the *connectEvent* method.

30.2.14.2. General Mouse Event Handler

There are a number of mouse events where the event handler receives the same arguments and the reply from the event handler is generally done in the same manner. These events are:

- LButtonDown
- LButtonUp
- LButtonDBlClk
- MButtonDown
- MButtonUp
- MButtonDBlClk
- MouseHover
- MouseMove
- RButtonDown
- RButtonUp
- RButtonDBlClk

The handler for these events is discussed here. When each event is generated is described in the arugment description for the *connectEvent* method.

The programmer decides whether to have the interpreter wait, or not wait, in the message processing loop for the reply from the event handler. This is done through the *connectEvent* method. The reply indicates if the event was processed or not. A common value to return would be **`.true`**.

```
::method onMouseEvent unguarded
  use arg state, pos, mouse

  return .true
```

Arguments:

The event handling method receives 3 arguments:

state

A list of keywords that indicate the keyboard and mouse button modifiers. The list will contain 1 or more of the following keywords separated by spaces. It will never be the empty string:

None	mButton	xButton1
Control	rButton	xButton2
lButton	Shift	

None
: There are no modifiers. *None* will always be the only keyword if it is present.

Control
: The control key is down.

lButton
: The left mouse button is down.

mButton
: The middle mouse button is down.

rButton
: The right mouse button is down.

xButton1
: The first X button is down.

xButton2
: The second X button is down.

pos
: A *Point* object that contains the x and y coordinates of the position of the mouse, relative to the upper-left corner of the screen.

mouse
: The Rexx *Mouse* object for the window that received the mouse event notification.

Return:

In general, return `.true` to indicate the event has been processed and `.false` to indicate it was not processed. However, the correct response is dependent on how the event was connected, and is summarized in the *Remarks* section for the *connectEvent* method.

Remarks:

Only windows that have the CS_DBLCLKS style can receive double click event notifications. The system generates the notification whenever the user presses, releases, and again presses the mouse button within the system's double-click time limit. Double-clicking a mouse button actually generates a sequence of four messages: mouse button down, mouse button up, mouse button double click, and mouse button up.

To receive the mouse hover event, the programmer must first invoke the *trackEvent* method with the proper arguments. The mouse tracking is canceled when the mouse hover notification is generated. To receive further mouse hover events, the *trackEvent* method must be invoked again.

Example

The following example uses the *connectEvent* method to connect the mouse left button down event sent to a list view window with the onLBDown method of the dialog.

```
::method initDialog
  expose listView

  listView = self~newListView(IDC_LV_DATA)

  mouse = .Mouse~new(listView)
  mouse~connectEvent(LBUTTONDOWN, onLBDown)

  ...
```

```
::method onLBdown unguarded
  expose listView
  use arg keyState, p, mouse

  index = listView~hitTestInfo(p)

  if keyState \== 'lButton' | index = -1 then do
    -- Pass the message on to the list view control.
    return .false
  end

  -- Since the list view will not receive the left button
  -- down event, we need to manually do the item selection
  listView~assignFocus
  listView~focus(index)
  listView~select(index)

  -- Do custom processing for left button down
  ...

  -- Indicate the event was processed.
  return .true
```

30.2.14.3. NcMouseHover Event Handler

The event handler for the nc mouse leave event is invoked when the cursor leaves the non-*client area* of the window assigned to the mouse object. Normally, this event notification is *not* generated. To receive the notification the *trackEvent* method must be invoked to set up the tracking behavior.

The programmer specifies if the interpreter waits or not for the return from the event handler through the arguments used in the *connectEvent* method.

```
::method onNcMouseHover unguarded
  use arg hit, pos, mouse

  return .true
```

Arguments:

The event handling method receives 3 argument:

hit

A single keyword that indicates the result of the operating system's hit test of the mouse position. This defines exactly where in the non-client area the mouse position is. Over the max button, on the border, etc.. The keyword will be one of the words in the *hit test table*.

pos

A *Point* object that contains the x and y coordinates of the position of the mouse, relative to the upper-left corner of the screen.

mouse

The Rexx *Mouse* object for the window that received the MOUSELEAVE or NCMOUSELEAVE event notification.

Return:

In general, return **`.true`** to indicate the event has been processed and **`.false`** to indicate it was not processed. However, the correct response is dependent on how the event was connected, and is summarized in the *Remarks* section for the *connectEvent* method.

Remarks:

To receive either of the mouse leave events, the programmer must first invoke the *trackEvent* method with the proper arguments. The mouse tracking is canceled when the mouse leave notification, (or the mouse hover notification,) is generated. To receive further mouse leave or mouse hover events, the *trackEvent* method must be invoked again.

30.2.14.4. MouseLeave Event Handler

This description for the mouse leave event handler pertains to both the *client area* mouse leave (MOUSELEAVE) and the *non-client* mouse leave (NCMOUSELEAVE) events. The arguments received by event handler for both events are exactly the same, and the event handler should behave the same for both events.

The event handler for the mouse leave event is invoked when the cursor leaves the client area (MOUSELEAVE) or the non-client area (NCMOUSELEAVE) of the window assigned to the mouse object. Normally, these event notifications are *not* generated. To receive these notifications the *trackEvent* method must be invoked to set up the tracking behavior.

The programmer specifies if the interpreter waits or not for the return from the event handler through the arguments used in the *connectEvent* method.

```
::method onMouseLeave unguarded
  use arg mouse

  return .true
```

Arguments:

The event handling method receives 1 argument:

mouse

The Rexx *Mouse* object for the window that received the MOUSELEAVE or NCMOUSELEAVE event notification.

Return:

In general, return **`.true`** to indicate the event has been processed and **`.false`** to indicate it was not processed. However, the correct response is dependent on how the event was connected, and is summarized in the *Remarks* section for the *connectEvent* method.

Remarks:

To receive either of the mouse leave events, the programmer must first invoke the *trackEvent* method with the proper arguments. The mouse tracking is canceled when the mouse leave notification is generated. To receive further mouse leave, the *trackEvent* method must be invoked again.

30.2.14.5. MouseWheel Event Handler

The event handler for the mouse wheel event is invoked each time the user scrolls the mouse wheel.

The programmer decides whether to have the interpreter wait, or not wait, in the message processing loop for the reply from the event handler. This is done through the *connectEvent* method. The reply indicates if the event was processed or not. A common value to return would be **`.true`**.

```
::method onMouseWheel unguarded
```

```
  use arg state, pos, mouse, delta

  return .true
```

Arguments:

The event handling method receives 4 arguments:

state

A list of keywords that indicate the keyboard and mouse button modifiers. The list will contain 1 or more of the following keywords separated by spaces. It will never be the empty string:

None	mButton	xButton1
Control	rButton	xButton2
lButton	Shift	

None

There are no modifiers. *None* will always be the only keyword if it is present.

Control

The control key is down.

lButton

The left mouse button is down.

mButton

The middle mouse button is down.

rButton

The right mouse button is down.

xButton1

The first X button is down.

xButton2

The second X button is down.

pos

A *Point* object that contains the x and y coordinates of the mouse position, relative to the upper-left corner of the screen.

mouse

The Rexx *Mouse* object for the window that received the MOUSEWHEEL event notification.

delta

A whole number that represents the amount the scroll wheel was turned. The operating system always sends a multiple of 120. Microsoft says this is to allow for future expansion with hardware that has a finer grain of control. If the number is positive, the user turned the wheel away from herself and indicates to scroll up. When negative the user turned the wheel towards himself, and indicates to scroll down.

At this point in time (c. 2011) the OS sends the event notification for each notch of the scroll wheel. Therefore delta is always 120 or -120.

Return:

In general, return **`.true`** to indicate the event has been processed and **`.false`** to indicate it was not processed. However, the correct response is dependent on how the event was connected, and is summarized in the *Remarks* section for the *connectEvent* method.

Remarks:

The mouse wheel event notification is sent by the operating system to the window with the current input focus. If the window does not process the message, then the OS sends the message to the parent window of that window. This continues on up the parent / child window chain until either a window does process the notification or the top of the chain is reached. Most dialog controls do not process the mouse wheel notification, so the notification will reach the dialog.

Multi-line edit controls *do* process the notification and normally handle their own scrolling. If the programmer wants to implement some type of custom scrolling using the *connectEvent* method, the edit control's interception of the notification may interfere with the custom scrolling. Forcing the edit control to ignore the notification allows it to reach the dialog.

Additional information on the mouse wheel and handling mouse wheel events is found in the system parameters information *SPI* class's *wheelScrollLines* attribute.

Example

The following example uses the *connectEvent* method to connect the mouse wheel event sent to the edit control to a method in the Rexx dialog. This then allows the programmer to supply some custom scrolling. Normally, a multi-line edit control handles the mouse wheel event itself and the notification message will never reach the dialog.

```
::method initDialog
  expose eData

  eData = self~newEdit(IDC_EDIT_DATA)

  mouse = .Mouse~new(eData)
  mouse~connectEvent(MOUSEWHEEL, onMouseWheel)

  ...

::method onMouseWheel unguarded
  expose eData
  use arg state, pos, mouse, delta

  -- If the mouse is not over the edit control, do nothing and
  -- return .true to indicate the event was processed.
  if \ pos~inRect(eData~windowRect) then return .true

  -- Mouse is over our edit control, do custom scrolling ...

  if delta > 0 then direction = 'up'
  else direction = 'down'

  scrollLines = .SPI~wheelScrollLines

  select
    when state == 'None' then do
      amt = scrollLines
      if direction == 'up' then cmd = 'UP'
      else cmd = 'DOWN'
    end

    when state~wordPos('Control') <> 0, state~wordPos('Shift') <> 0 then do
      amt = 3
      if direction == 'up' then cmd = 'PAGEUP'
      else cmd = 'PAGEDOWN'
    end

    when state~wordPos('Control') <> 0 then do
      amt = 1
      if direction == 'up' then cmd = 'PAGEUP'
      else cmd = 'PAGEDOWN'
    end

    when state~wordPos('Shift') <> 0 then do
      amt = 3 * scrollLines
      if direction == 'up' then cmd = 'UP'
      else cmd = 'DOWN'
    end

    otherwise do
      -- Some other modifier(s), we treat this
      -- the same as 'None'
      amt = scrollLines
      if direction == 'up' then cmd = 'UP'
      else cmd = 'DOWN'
    end
  end

  eData~scrollCommand(cmd, amt)

  -- Indicate the event was processed.
  return .true
```

30.2.15. cross

```
>>--cross----------------------------------------><
```

Sets the mouse cursor to the system's default *high precision* cursor, which is usually the crosshair cursor.

Arguments:

This method takes no arguments.

Return value:

Returns the previous cursor on success, or 0 on error.

Remarks:

The *cross* method is convenience method. It is exactly equivalent to using the *setCursor* method with the CROSS keyword.

Details

Sets the *.systemErrorCode*.

Example:

This a simple method to use:

```
mouse = .Mouse~new(self)
oldCursor = mouse~cross
```

30.2.16. dragDetect

```
>>--dragDetect(--point--)------------------------><
```

The *dragDetect* method is used to determine if the user has started a drag operation.

Arguments:

The single argument is:

point [required]

The initial position of the cursor, in screen *coordinates*) coordinates, specified as a *Point* object. Recall that screen coordinates are always in pixels.

Return value:

The return is true if the user is dragging and false otherwise.

Remarks:

In a drag and drop enabled program it is not desirable to start a drag operation each time the user presses the left mouse button. This results in the user moving something when they only wanted to click on it. The operating system provides the drag detect functionality as a convenience to programmers.

A drag operation is detected when the user presses the mouse button and then moves the cursor outside of the *drag rectangle*, while keeping the mouse button down. The drag rectangle is configurable, allowing for smaller or larger rectangles. The size of the current drag rectangle, and changing the size of the drag rectangle can be done through the *dragWidth* and *dragHeight* attributes of the *SPI* (System Parameters Information) class.

Details

Raises syntax errors when incorrect usage is detected.

Resets the *.systemErrorCode* to 0. However, there are no system errors that would change that value.

Example:

This example shows the event handler for a left button down event. When the event occurs, the handler checks if the user has started a drag operation. If so, the application initiates the operation and returns true. If not, the handler simply returns:

```
::method onLBdown unguarded
  expose dragItem
  use arg keyState, p, mouse

  if mouse~dragDetect(p) then do
    dragItem = self~doDrag(p, mouse)
    return .true
  end

  return .false
```

30.2.17. getCapture

```
>>--getCapture-----------------------------------><
```

The *getCapture* method retrieves the window handle, if any, that currently has captured the mouse. The window that has captured the mouse receives all mouse input regardless of whether the mouse cursor is within the borders of the window or not.

Arguments:

This method has no arguments.

Return value:

The window *handle* of the window, belonging to the same dialog as this mouse's window, that had previously captured the mouse, or 0 if no window previously had the capture.

Remarks:

The mouse capture functions work with windows of the same thread. For ooDialog, this means the dialog window and the dialog control windows that belong to the same dialog as this mouse's window. Getting the mouse capture, can only get the window handle of windows belonging to the same dialog as this mouse's window. If 0 is returned, it means that no window in the same dialog as this mouse's window has the mouse capture. However, some window belonging to another dialog could have the mouse capture.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode* to 0, but there are no system errors that would change the variable from 0.

30.2.18. getClipCursor

```
>>--getClipCursor(--rect--)----------------------><
```

Retrieves the screen coordinates of the rectangular area to which the cursor is confined.

Arguments:

The single argument is:

rect [required]

A *Rect* object in which the coordinates are returned.

Return value:

True on success, false on error.

Remarks:

If the cursor is not confined, i.e., it is *clipped*/>, then on return *rect* will contain the coordinates of the screen.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

30.2.19. getCursorPos

```
>>--getCursorPos---------------------------------><
```

Retrieves the current position of the cursor.

Arguments:

There are no arguments.

Return value:

Returns the postion, in screen *coordinates*, as a *Point* object.

Details

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect usage is detected.

Example:

The setCursorPos *example* uses the *getCursorPos* method.

30.2.20. isButtonDown

```
>>--isMouseButtonDown(--+---------------+--)-----><
                        +--whichButton--+
```

Retrieves information on whether a mouse button is pressed.

Arguments:

The single optional argument is:

whichButton [optional]

If this argument is omitted the default keyword is LEFT. Otherwise, use exactly one of the following keywords, case is not significant:

LEFT	MIDDLE	XBUTTON2
RIGHT	XBUTTON1	

LEFT

Determine if the primary mouse button is down. The primary mouse button may actually be the right physical mouse button if the user has configured the system to swap the mouse buttons.

RIGHT

Determine if the secondary mouse button is down. The secondary mouse button may actually be the left physical mouse button if the user has configured the system to swap the mouse buttons.

MIDDLE

Determine if the middle mouse button is down.

XBUTTON1

Determine if the first X button is down. XBUTTON1 and XBUTTON2 are used by the operating system to support the Microsoft IntelliMouse Explorer.

XBUTTON2

Determine if the second X button is down. XBUTTON1 and XBUTTON2 are used by the operating system to support the Microsoft IntelliMouse Explorer.

Return value:

Returns true if the specified button is being pressed, otherwise false.

Details

Raises syntax errors if incorrect usage is detected.

Sets the *.systemErrorCode* to 0, but there are no operarting system errors that would change it from 0.

Example:

This example carries out some operation, does stuff, while either the right or the left mouse button, but not both buttons, is pressed:

```
lMB = self~isMouseButtonDown("LEFT")
rMB = self~isMouseButtonDown("RIGHT")
do while (lMB | rMB) & \(lMB & rMB)
    -- do stuff
    ...

    lMB = self~isMouseButtonDown("LEFT")
    rMB = self~isMouseButtonDown("RIGHT")
end
```

30.2.21. no

```
>>--no-------------------------------------------><
```

Sets the mouse cursor to the system's default *not allowed* cursor, which is usually a circle with a slash across it.

Arguments:

This method takes no arguments.

Return value:

Returns the previous cursor on success, or 0 on error.

Remarks:

The *no* method is convenience method. It is exactly equivalent to using the *setCursor* method with the NO keyword.

Details

Sets the *.systemErrorCode*.

Example:

This a simple method to use:

```
mouse = .Mouse~new(self)
oldCursor = mouse~no
```

30.2.22. releaseCapture

```
>>--releaseMouseCapture--------------------------><
```

The *releaseCapture* method releases the mouse capture from the window, belonging to the same dialog as this mouse's window, which had previously *capture* the mouse. Normal mouse input processing is then restored.

Arguments:

There are no arguments for this method.

Return value:

Returns 0 on success, the mouse capture was released, or 1 on failure. On failure, the **`.systemErrorCode`** will be set.

Remarks:

The mouse capture functions work with windows of the same thread. For ooDialog, this means the dialog window and the dialog control windows that belong to the same dialog as this mouse's window.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

Example:

This example is a snippet of code from a drag and drop enabled program. It is part of the processing of the left button up event handler. If *dragging* is **`.true`** then we know that the mouse was captured when the drag operation was initiated, so the capture is released. The return from *releaseCapture* is not checked because it is unlikely to fail:

```
if dragging then do
  okayToDrop = (cursorIsNoDrop \== .true)

  dragging = .false
  cursorIsNoDrop = .false
  mouse~releaseCapture
  mouse~setCursor(oldCursor)
  ...
end
```

30.2.23. releaseClipCursor

```
>>--releaseClipCursor----------------------------><
```

Releases the cursor so that it is free to move anywhere on the screen.

Arguments:

There are no arguments for this method.

Return value:

Returns true on success, false on error.

Remarks:

The cursor is a shared resource. If an application confines the cursor through the *clipCursor* method, it must release the cursor at some point by using *releaseClipCursor* before the user can move to another application using the mouse.

Although the MSDN documentation does not explicity state this, experimentation shows that if the user brings up the Alt-Tab switching dialog and moves to another application in that manner, the cursor is no longer confined. If the user closes the dialog, the mouse is no longer confined.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

Example:

This example releases the cursor from the clipped state.

```
mouse = .Mouse~new(self)
mouse~releaseClipCursor
```

30.2.24. restoreCursor

```
>>--restoreCursor(--+----------+--)--------------><
                    +--cursor--+
```

Restores the cursor. This is a convenience method, the same functionality can be achieved using the *setCursor* method.

Arguments:

The single argument is:

cursor [optional]

A cursor *Image* object that will be set as the cursor for the window of this mouse instance. If this argument is omitted then the operating system's pre-defined *normal* cursor, which is usually the arrow cursor, is set as the cursor for the window.

Return value:

Returns the previous cursor on success, or 0 on error.

Remarks:

Normal usage of this method would generally be to use a cursor saved using the return from the *setCursor* method as the *cursor* argument. Assuming the previous cursor was the arrow cursor and omitting the *cursor* argument can be wrong if the user has customized his system.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

Example:

This example calculates pi to 35 decimal places. While the calculation is being done the cursor is changed to the busy cursor, and then restored when the calculation is finished:

```
-- Get the current mouse position.
mouse = .Mouse~new(self)
oldCursorPos = mouse~getCursorPos

-- Set the cursor to the busy symbol and jiggle the
-- mouse so it repaints.
oldCursor = mouse~wait
oldCursorPos~incr
mouse~setCursorPos(oldCursorPos)

-- Calculate pi to 35 decimal places.
pi = self~calcPi(35)

-- Restore the cursor.
oldCursorPos~decr
mouse~restoreCursor(oldCursor)
mouse~setCursorPos(oldCursorPos)
```

30.2.25. setCursor

```
>>--setCursor(--cursor--)------------------------><
```

Sets the cursor for the window of this mouse instance.

Arguments:

The single argument is:

cursor [required]

This argument can either be a *Image* cursor object, in which case the image is used for the new cursor, or a single keyword. When *cursor* is a keyword, the new cursor is set to one of the shared pre-defined system cursors.

When using a keyword as the *cursor* argument, it must be exactly one of the following to indicate which system cursor is desired, case is not significant:

APPSTARTING	IBEAM	SIZENWSE
ARROW	NO	SIZEWE
CROSS	SIZEALL	UPARROW
HAND	SIZENESW	WAIT
HELP	SIZENS	

APPSTARTING
: Standard arrow and small hourglass.

ARROW
: Standard arrow.

CROSS
: Crosshair.

HAND
: Hand.

HELP
: Arrow and question mark.

IBEAM
: I-beam.

NO
: Slashed circle.

SIZEALL
: Four-pointed arrow pointing north, south, east, and west.

SIZENESW
: Double-pointed arrow pointing northeast and southwest.

SIZENS
: Double-pointed arrow pointing north and south.

SIZENWSE
: Double-pointed arrow pointing northwest and southeast.

SIZEWE
: Double-pointed arrow pointing west and east.

UPARROW
: Vertical arrow.

WAIT
: Hourglass.

Return value:

On success, returns the previous cursor for the window as an **`Image`** object. On error returns 0.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

Example:

This example is similar to the example for the *restoreCursor* method, but it only uses the *setCursor* method:

```
-- Get the current mouse position.
mouse = .Mouse~new(self)
oldCursorPos = mouse~getCursorPos

-- Set the cursor to the busy symbol and jiggle the
-- mouse so it repaints.
oldCursorPos~incr
oldCursor = mouse~setCursor("WAIT")
mouse~setCursorPos(oldCursorPos)

-- Calculate pi to 35 decimal places.
pi = self~calcPi(35)

-- Restore the cursor.
oldCursorPos~decr
mouse~setCursor(oldCursor)
mouse~setCursorPos(oldCursorPos)
```

30.2.26. setCursorPos

```
Form 1:

>>--setCursorPos(--point--)----------------------><

Form 2:

>>--setCursorPos(--x--,--y--)--------------------><

Generic form:

>>--setCursorPos(--newPosition--)----------------><
```

The *setCursorPos* method moves the mouse cursor to the specified position. This method can be used to force the repainting of the mouse cursor or to keep the mouse cursor within a specific rectangle.

Arguments:

The argument(s) specify the (x, y) coordinates for the new position of the mouse cursor. These coordinates can be specifed either as a *Point* object or as 2 separate whole number arguments, as in Form 2.

newPosition [required]

Whether the coordinates are specified as a **`point`** object or in the x, y format both coordinates are required. The coordinates are specified as screen *coordinates*) coordinates.

Return value:

Returns 0 on success or 1 on failure.

Details

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect arguments are detected.

Example:

The following example shows two methods: one indicating that processing has started and one indicating that processing has completed. The method *indicateBeginProcessing* changes the mouse cursor to the WAIT cursor and *indicateEndProcessing* restores the original mouse cursor. Both methods retrieve the current position of the mouse and move it by one screen pixel in each direction to force the repainting of the cursor.

```
::method indicateBeginProcessing
  expose oldCursor

  mouse = .Mouse~new(self)
  oldCursor = mouse~wait
  p = mouse~getCursorPos
  mouse~setCursorPos(p~incr)

::method indicateEndProcessing
  expose oldCursor

  mouse = .Mouse~new(self)
  mouse~restoreCursor(oldCursor)
  p = mouse~getCursorPos
  mouse~setCursorPos(p~decr)
```

30.2.27. showCursor

```
>>--showCursor(--+--------+--)-------------------><
                 +--show--+
```

Displays or hides the cursor.

Arguments:

The single argument is:

show [optional]

Specifies whether the internal display counter, (see the Remarks section,) is to be incremented or decremented. If .true, the display count is incremented by one. If **`.false`**, the display count is decremented by one. The default is **`.true`**.

Return value:

The new display count.

Remarks:

The operating system maintains an internal display counter that determines whether the cursor should be displayed. The cursor is displayed only if the display count is greater than or equal to 0. If a mouse is installed, the initial display count is set to 0. If no mouse is installed, the display count is -1.

In early versions of Windows the internal counter was system wide, but now it is thread specific. What this means in terms of ooDialog is that the cursor state, shown or hidden, will apply only to the dialog and all its child windows. For a dialog, the child windows are the dialog control windows, and any *ControlDialog* windows, if present.

Note that this method does not hide or show the cursor, it changes the internal counter. This implies that if the programmer invokes this method 10 times with *show* equal to **`.false`**, it will take at least 10 invocations of the method with *show* equal to **`.true`** to get the cursor to show.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode* to 0. However, there are no system errors that would change the variable from 0.

Example:

This example is a code snippet from a program that hides the cursor while it is over one of the dialog windows in the program and shows it when not over that dialog window

```
::method hideCursor private
  use strict arg obj

  mouse = .Mouse~new(obj)
  do while mouse~showCursor(.false) >= 0
    nop
  end

::method showCursor private
  use strict arg obj

  mouse = .Mouse~new(obj)
  do while mouse~showCursor(.true) < 0
    nop
  end
```

30.2.28. trackEvent

```
>>--trackEvent(--+--------+--+-------------+--+----------+--)------------------><
                 +-event-+  +-,-hoverTime-+  +-,-answer-+
```

The *trackEvent* method initiates mouse tracking behavior. It can also be used to query the current tracking behavior. When mouse tracking is in effect mouse hover and mouse leave *event* notifications, depending on the type of tracking specified, will be generated. Normally these event notifications are *not* generated.

Arguments:

The arguments are:

event [optional]

A list of 1 or more of the following keywords separated by spaces, case is not significant. The keywords define the tracking behavior to initiate. If this argument is omitted, the default list is *LEAVE*:

CANCEL	LEAVE	QUERY
HOVER	NONCLIENT	

CANCEL

Cancel a prior tracking request. The type of tracking to be canceled should also be specified. For example, to cancel hover tracking, the keyword list should be *Cancel Hover*. (Case is not significant of course.)

HOVER

Hover notifications should be generated. The notification generated is the MOUSEHOVER notification, connect this event using the *connectEvent*method.

If hover tracking is requested while hover tracking is already active, the hover timer will be reset. The operating system completely ignores this request if the mouse pointer is not over the window of this mouse instance.

LEAVE

Leave notifications should be generated. The notification generated is the MOUSELEAVE notification, connect this event using the *connectEvent*method. If the mouse is not over the window of this mouse instance, a leave notification is generated immediately and no further tracking is performed.

NONCLIENT

Non-*client area* hover and leave notifications should be generated. The notifications generated are the NCMOUSELEAVE and NCMOUSEHOVER notifications, connect these events using the *connectEvent*method.

QUERY

Rather than treating this as a tracking request, the *answer* argument, which must be a **`.Directory`** object, is filled in. The **`Directory`** object will contain an EVENT and a HOVERTIME index. These indexes will contain values such that, had they been used for the *event* and *hoverTime* arguments to the *trackEvent* method, the method would generate the current tracking.

The only anomaly is that the hover time out returned is always the actual time out and not *HOVER_DEFAULT*, if HOVER_DEFAULT was specified as *hoverTime* during the original *trackEvent* request.

hoverTime [optional]

Specifies the hover time out, (if HOVER was specified in *event*,) in milliseconds. The constant, *HOVER_DEFAULT* can also be use to specify the system default hover time out. The value of the system default hover time out can be determined, or changed, through the *mouseHoverTime* attribute of the *SPI* (System Parameters Infomation) class.

answer [optional] [in / out]

A **`Directory`** object in which the results of a QUERY request are returned. This is required if the *event* argument contains the QUERY keyword, otherwise this argument is ignored. On a successful return the **`Directory`** object will contain the following indexes:

EVENT

A keyword list, using the same keywords that are listed for the *event* argument. The keywords will reflect the current tracking. If the list had been used in a tracking request, they would produce the current tracking.

HOVERTIME

The hover time out in effect for the current tracking. However, if HOVER_DEFAULT had been used for the *hoverTime* argument in the tracking request, this value will be the actual time out value, not the value of HOVER_DEFAULT.

Return value:

Returns true on success, otherwise false.

Remarks:

The mouse pointer is considered to be *hovering* when it stays within a specific rectangle for a specific period of time. The *SPI* class has attributes that can be used to retrieve (or change) the hover rectangle or hover time out value. The *mouseHoverWidth* and *mouseHoverHeight* attributes reflect the hover rectangle. The *mouseHoverTime* attribute reflects the hover time out.

The system default hover time out is initially the menu drop-down time, which is 400 milliseconds. The system default hover rectangle is the same as the double-click rectangle. However, these values may be changed / customized by the user.

Details

Raises syntax errors when incorrect arguments are detected.

Sets the *.systemErrorCode*.

Example:

This example connects the mouse move, leave, and hover events. The first time a mouse move is received, the *trackEvent* method is used to request tracking for the leave and hover events. When either one of the events is received, the tracking for that event is canceled. In order to receive further events, a flag is set so that on the next mouse move event, the *trackEvent* method is invoked again.

```
::method init

  ...

  mouse = .Mouse~new(self)
  mouse~connectEvent('MOUSEMOVE')
  mouse~connectEvent('MOUSELEAVE')
  mouse~connectEvent('MOUSEHOVER')

  -- so leave and hover tracking is requested on first mouse move
  mouseEntered  = .false
  mouseHovering = .true

::method onMouseMove unguarded
  expose mouseEntered mouseHovering
  use arg keyState, p, mouse

  if \ mouseEntered then do
    if \ mouse~trackEvent("LEAVE") then do
      say "trackEvent() failed. Error" .systemErrorCode ':'
 SysGetErrorText(.systemErrorCode)
    end
    mouseEntered = .true
  end
  if mouseHovering then do
    if \ mouse~trackEvent("HOVER") then do
      say "trackEvent() failed. Error" .systemErrorCode ':'
 SysGetErrorText(.systemErrorCode)
    end
    mouseHovering = .false
  end
  ...
  return .true

::method onMouseLeave unguarded
  expose mouseEntered
  use arg mouse

  mouseEntered = .false

  -- so hover tracking is started when mouse re-enters.
  mouseHovering = .true
  ...
  return .true

::method onMouseHover unguarded
  expose mouseHovering
  use arg keyState, mousePos, mouse

  mouseHovering = .true
  ...
  return .true
```

30.2.29. wait

```
>>--wait-----------------------------------------><
```

Sets the mouse cursor to the system's default *busy* cursor, which is usually the hourglass cursor.

Arguments:

This method takes no arguments.

Return value:

Returns the previous cursor on success, or 0 on error.

Remarks:

The *wait* method is convenience method. It is exactly equivalent to using the *setCursor* method with the WAIT keyword.

Details

Sets the *.systemErrorCode*.

Example:

This a simple method to use:

```
mouse = .Mouse~new(self)
oldCursor = mouse~wait
```

30.3. Keyboard Class

The **`Keyboard`** class has methods and attributes to address the functionality of receiving and processing keyboard input as supplied by the Windows operating system.

Currently the **`Keyboard`** class has few methods. It is anticipated that the class will be enhanced in future versions of ooDialog.

30.3.1. Method Table

The following table lists the class and instance methods of the **`Keyboard`** class:

Table 30.11. Keyboard Class Method Reference

Method	Description
Class Methods	**Class Methods**
getAsyncKeyState	Determines whether a key is up or down at the time this method is invoked. Optionally, determines whether the key was pressed after a previous invocation of the *getAsyncKeyState* method.
Instance Methods	**Instance Methods**

30.3.2. getAsyncKeyState (Class Method)

```
>>--getAsyncKeyState(--vKey--+---------+--)------><
                             +-,-info--+
```

Determines whether a key is up or down at the time this method is invoked. Optionally, determines whether the key was pressed after a previous invocation of the *getAsyncKeyState* method.

Arguments:

The arguments are:

vKey [required]

The virtual key code of the key being queried. It makes the most sense to us the *VK* class to supply the key code, however, all that is needed is the correct numeric value of the virtual key.

info [optional in / out]

A **`Directory`** object whose indexes will, on return, contain the key state information. On return the **`Directory`** will contain the following indexes:

WASDOWN

This index will be true if the key was down and false if the key was not down. The value of this index is exactly the same as the return from the *getAsyncKeyState* method.

WASPRESSED

This index reflects the state of the operating system's *was pressed* flag. That flag is true if the key was pressed since the last time the async key state was queried. The *WASPRESSED* index will be true if the flag for the specified key was true and false if the flag is false. Microsoft cautions that this is not reliable, see the remarks.

Return value:

Returns true if the specified virtual key was down and false if it was not down.

Remarks:

Microsoft cautions that the *was pressed* state is not reliable. The reason is this, the state is a system wide state. Since modern versions of Windows use pre-emptive multitasking, if another application queries the async state, it could receive the *was pressed* state rather than the application currently querying the state. Microsoft maintains the state flag for compatibility with early versions of Windows which were not pre-emptive. ooDialog allows the optional querying of the *was pressed* state for completeness, and the idea that presumably the programmer knows what she is doing.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example comes from an application that has a user interface that does one thing if the user clicks a column header of a list-view, and another thing if the column click was done with the control key held down:

```
::method onColumnClick unguarded
  use arg id, colIndex, listView

  if colIndex == 2 & .Keyboard~getAsyncKeyState(.VK~CONTROL) then do
    self~commitLastNames
    return 0
  end

  ...

  return 0
```

Utility Classes, Routines, and Objects

The ooDialog framework provides a number of utility classes and public routines that are useful when writing more complex programs. These classes and routines do not easily fit into the category of a type of dialog or a type of dialog control and are therefore documented separately. In addition, ooDialog places some globally available objects in the `.local` environment.

The classes, routines, and objects in the following table are described in this chapter:

Table 31.1. ooDialog Utility Classes, Routines, and Objects

Utility Class, Routine, or Object	Description
.application	An instance of the `ApplicationManager` class that is present in all ooDialog programs.
ApplicationManager	Manages application (global) settings for a single ooDialog program.
.constdir	an ooRexx `.Directory` object used to resolve *symbolic* IDs.
DayState	Represents the state of each day within a single month. A helper class for the *MonthCalendar*.
DayStates	A sequential collection of day state objects.
DlgArea	Assists in laying out the dialog controls in a dynamically defined dialog.
DlgAreaU	Assists in the creation of dynamically resizable dialogs.
DlgUtil	Provides some common, useful, utilities for working with dialogs.
FindWindow	Searches the desktop for a specific window and returns its handle.
Locate	Returns the directory containing the Rexx source code file that *Locate* is executing in. Similar to *parse source*.
MSSleep	Sleeps for the specified number of milliseconds.
OS	Provides methods for accessing information about the operating system.
Play	Used to play audio sounds.
Point	An object that represents a point in a 2-D coordinate system.
Rect	An object used to represent a rectangle in a 2-D coordinate system.
ScreenSize	Obtains the screen size in dialog units and in pixels.
Size	An object encapsulating a width and height dimension in a 2-D coordinate system.
SM	A class whose attributes reflect the system metrics or configuration settings of the computer.
SPI	A class whose attributes reflect the system-wide parameters of the computer.
.systemErrorCode	Reflects the *system error code*, if set.
VK	Translates back and forth between virtual key code numbers and symbolic names.
Window	An utility class with methods common to all windows.
WinTimer	Depending on the specified mode, creates, waits on, or releases a periodic Windows timer.

31.1. .application object

The **`.application`** object is an instance of the *ApplicationManager* class that is present in all ooDialog programs. The object is initialized and placed in the **`.local`** environment when the ooDialog package is loaded.

The **`.application`** object is used by the programmer to manage application wide settings and constants. Its use is primarily to change global default values of ooDialog. For instance, by default, *automatic* data detection is on. (Automatic data detection is part of the *data* attributes concept.) The **`.application`** object can be used to change the default for all dialogs in a application to off.

It would be most common to use the **`.application`** object at the start of a program. This example shows a typical way of changing the default for automatic data detection off for the entire application:

```
.application~autoDetection(.false)

dlg = .SimpleDialog~new("UserMenuBar.rc", IDD_MAIN_DIALOG)
if dlg~initCode <> 0 then do
  return 99
end

dlg~execute("SHOWTOP")
...
```

31.2. ApplicationManager Class

An application manager is used to set certain application wide defaults and to perform tasks that effect the entire ooDialog application. Examples of application wide defaults are the default *font* and font *size*, whether *automatic* data detection is on or off, etc.. An example of a task that effects the entire application is the populating of the *.constdir* with symbols.

When an ooDialog application starts, the ooDialog framework instantiates an **`ApplicationManager`** object and places it in the **`.local`** environment as the *.application*. Only one instance of an **`ApplicationManager`** per application can be instantiated. In practice this means that the ooRexx programmer can not create a new **`ApplicationManager`** object. The programmer invokes all methods of the **`ApplicationManager`** through the **`.application`** object.

31.2.1. Method Table

The following table lists the class and instance methods of the **`ApplicationManager`** class:

Table 31.2. ApplicationManager Class Method Reference

Method	Description
Class Methods	**Class Methods**
new	A new application manager can not be instantiated in Rexx code.
Attributes	**Attributes**
Section 31.2.3, “constDir (Attribute)”	A directory object that maps symbolic resource IDS to their numeric IDs
srcDir	Reflects the complete path name of the directory the main program file is located in, provided the *Locate* method has been executed in the interpreter process.

Method	Description
Instance Methods	
addToConstDir	Adds symbols, symbolic IDs, to the **`.constDir`**.
autoDetection	Sets the application global default for *autoDetection* to off, or on.
defaultFont	Changes the *default* font for the application.
defaultIcon	Sets a default icon to be used for the *application* icon for all dialogs that do not explicitly set their own application icon.
parseIncludeFile	Reads a file and adds any symbol definitions found to the proper constant directory.
resolveNumericID	Resolves a numeric ID to a symbolic ID.
resolveSymbolicID	Resolves a symbolic ID to its numeric value.
requiredOS	Checks that an ooDialog program is executing on a required minimum Windows version.
setDefaults	Sets one or more of the global defaults for the application.
useGlobalConstDir	Specifies how the global *.constdir* is used in the application.

31.2.2. new

```
>>--new(--opaque--)-----------------------------><
```

A new instance of the **`ApplicationManager`** can not be instantiated from Rexx code. A single instance for the **`ApplicationManager`** is instantiated by the ooDialog framework for each application and placed in the **`.local`** environment as the *.application*. All methods of the **`ApplicationManager`** are invoked by the programmer though the **`.application`** object.

31.2.3. constDir (Attribute)

```
ResourceUtils::constDir


>>--constDir------------------------------------><

>>--constDir[symbol]=numericValue---------------><
```

31.2.4. srcDir (Attribute)

```
>>--srcDir----------------------------------------------------><

>>--srcDir = varName------------------------------------------><
```

Reflects the complete path name of the directory the main program file is located in, provided the *Locate* routine has been executed in the interpreter process. The path name includes the trailing slash (\.)

srcDir get:

Returns a complete path name of a directory, or the **`.nil`** object if the *Locate* routine has not been executed.

srcDir set:

The programmer can not set the value of this attribute directly. The ooDialog framework sets, or resets, its value when the *Locate* routine is executed.

Remarks:

The purpose of the *Locate* function and the *srcDir* attribute is to help the programmer create complete path names for any auxiliary files needed by an application. This allows the application to execute correctly, even if started from a directory other than its installation directory. The documentation for the *Locate* routine contains some additional insight into this purpose.

Note that the intention for the *srcDir* attribute is to return the complete directory path name of the main application's source directory. However, this is dependent on the programmer's usage of the *Locate* routine. The actual value of the attribute is the directory in which the source code file, in which *Locate* is executed. If the programmer calls *Locate* in a source code file that is not located in the application's main directory, then the *srcDir* attribute is going to that directory.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows the *Locate* function being used to get the installation directory of an application and then the *srcDir* attribute being used to access the directory value in a private method of a class:

```
  srcDir = Locate()

  rcFile = srcDir"resources\imageButton.rc"
  symbolFile = srcDir"resources\imageButton.h"

  .application~setDefaults("O", symbolFile, .false)

  dlg = .ImageListDlg~new(rcFile, IDD_IMAGELIST_BUTTON)
  ...

::class 'ImageListDlg' subclass RcDialog
  ...

::method setPictureButtons private
  expose pbView pbAdd stStatus imagesLoaded imageList

  if .DlgUtil~comCtl32Version < 6 then return self~oldSetButtons

  pbView~style = "MULTILINE BOTTOM"

  srcDir = .application~srcDir

  -- The images are loaded from files.
  files = .array~new()
  files[1] = srcDir"resources\Normal.bmp"       -- Normal
  files[2] = srcDir"resources\Hot.bmp"          -- Hot (hover)
  files[3] = srcDir"resources\Pushed.bmp"       -- Pushed
  files[4] = srcDir"resources\Disabled.bmp"     -- Disabled
  files[5] = srcDir"resources\Default.bmp"      -- Default button
  files[6] = srcDir"resources\Hot.bmp"          -- Stylus hot, tablet PC only
  ...
```

31.2.5. addToConstDir

```
>>--addToConstDir(--symbols--)-------------------><
```

The *addToConstDir* is used to add symbolic IDs to the global *.constdir* object. The *useGlobalConstDir*, *parseIncludeFile*, and *setDefaults* methods can also be used to add symbols to the **`.constDir`**.

Arguments:

The arguments are:

symbols [required]

symbols can either be the name of a file containing symbol *definitions*s, or a collection object that maps *symbolic* IDs to their numeric value.

Return value:

True on success, false on error.

Remarks:

If *symbols* is a string it is taken to be the name of a file containing symbol definitons. The result in this case is the same as using the *parseIncludeFile*. The file is parsed and any symbol definitions are added to the **`.constDir`**.

When *symbols* is not a string, it must be an object with a *supplier* method where the indexes the supplier provides are strings. The items the supplier provides are added to the **`.constDir`** using the supplied indexes.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example creates a table of symbolic IDs and then uses that table to add symbols to the **`.constDir`**

```
  symbols = getSymbols()
  .application~addToConstDir(symbols)

  ...

::routine getSymbols

  tbl = .Table~new
  tbl[IDD_CONTEXT      ] = 1000
  tbl[IDC_LV           ] = 1003
  tbl[IDC_ST_MSG       ] = 1004

  tbl[IDM_RC_DLG       ] = 10000
  tbl[IDM_RC_SHOW      ] = 40000
  tbl[IDM_RC_ENABLE_LV ] = 40005
  tbl[IDM_RC_DISABLE_LV] = 40001
  tbl[IDM_RC_BEEP      ] = 40002
  tbl[IDM_RC_QUIT      ] = 40003

  tbl[IDM_LV_BAR       ] = 20000
  tbl[IDM_LV_SORT      ] = 20001
  tbl[IDM_LV_JUMBLE    ] = 20002
  tbl[IDM_LV_REVERSE   ] = 20003
  tbl[IDM_LV_SEP1      ] = 20004
  tbl[IDM_LV_SEP2      ] = 20005
  tbl[IDM_LV_FNAME     ] = 20006
```

```
    tbl[IDM_LV_LNAME     ] = 20007
    tbl[IDM_LV_PROFESSION] = 20008
    tbl[IDM_LV_RESTORE   ] = 20009

    return tbl
```

31.2.6. autoDetection

```
>>--autoDetection(--+------+--)------------------><
                    +--on--+
```

The *autoDetection* method is used to turn the application-wide default for *initAutoDetection* data field detection on or off. When an ooDialog program starts, the default for automatic data field detection is on.

Arguments:

The single argument is:

on [optional]

If *on* is true, the default for automatic data detection is set to on, if false it is set to off. If *on* is omitted automatic data detection is turned off.

Return value:

This method always returns 0.

Remarks:

Automatic data detection can also be turned off using the *setDefaults* method.

Details

Raises syntax errors when incorrect arguments are detected.

31.2.7. defaultFont

```
>>--defaultFont(--+-----------+--+------------+--)-----------><
                  +--fontName--+  +-,-fontSize--+
```

Sets the application-wide default dialog font.

Arguments:

The arguments are:

fontName [optional]

The name of the new default font for the dialog. If omitted the global font name is not changed. However, if both name and size are omitted the font is set back to the default font and size. (See remarks.)

fontSize [optional]

The new default size for the dialog font. If omitted the global font size is not changed. However, if both name and size are omitted the font is set back to the default font and size. (See remarks.)

Return value:

Returns true on success, otherwise false.

Remarks:

The programmer can change the global font back to the default font and font size by omitting both arguments. The *setDefaultFont* class method of the *dialog* object can also be used to set the application-wide default dialog font.

The length of the font name must be less than 256 characters and can not be the empty string, otherwise a syntax condition is raised. This is the only possible cause of failure. Note that since a syntax condition is raised on error, false will never actually be returned.

Details

Raises syntax errors when incorrect arguments are detected.

31.2.8. defaultIcon

```
>>--defaultIcon(--src--+------+--+---------+--)---------------><
                       +-,-id-+  +-,-binary-+
```

Sets a default icon to be used as the *application* icon for all dialogs that do not explicitly set their own application icon.

Arguments:

The arguments are:

src [required]

Indicates the source of the icon. The other arguments are used, or not used, to specify exactly what the *src* is. The source for the icon can be one of the following:

ResourceImage

If *src* is a *ResourceImage* object, then the *id* argument must be used to specify the resource ID of the icon in the resource image. The *binary* argument will be ignored in this case because a resource image is known to be binary.

File name of a text resource script

When *src* is the file name of a resource *script* file, the *id* argument must be used to specify the ICON resource ID to be used. The *binary* argument must be false to distinguish that the file is not a compiled resource file. The resource script will be searched for an ICON statement matching the *id* argument value and loaded if found.

File name of a binary compiled resource.

If *src* is the name of a binary file containing the icon resource, then the *id* argument must specify the resource ID of the icon. In addition, the *binary* argument must be set to true. This is the only way to distinguish that the file is a compile binary file and not a text resource script file.

Stand alone icon file

The *src* argument can be the file name of a stand alone icon file. In this case the *id* argument must be omitted. When *src* is not a **`ResourceImage`** and *id* is omitted, then the ooDialog framework assumes *src* is the name of an icon file, and the *binary* argument is ignored.

oodialog.dll

If *src* is exactly *oodialog.dll*, case insignificant, then the icon is loaded from the ooDialog DLL. ooDialog embeds a few icon images in its DLL file for *general* use by the programmer. If the *id* argument is not omitted, it must specify one of the pre-defined resource IDs of those icons. If *id* is ommitted then the default ID, IDI_DLG_OODIALOG, is used. This has no effect, unless the application has previously changed the default icon and now wants to revert to the default. The *binary* argument is ignored in this case because it is known that oodialog.dll is binary.

id [optional]
: The resource ID of the icon in the *src*. May be numeric or *symbolic*. This argument must be omitted if *src* is the name of an icon file, and may be omitted if *src* is *oodialog.dll*. In all other cases it is required.

binary [optional]
: True to specify that *src* is a binary compiled file and false to specify that *src* is a resource script file. The defualt is false.

Return value:

Returns true on success, otherwise false. In most cases of failure, the **`.SystemErrorCode`** object will contain the system error number for the failure.

Remarks:

The *application*, (or *dialog*,) icon can be explicitly set using an argument to the *execute* method, or any of the other methods that start a dialog executing such as the *executeAsync*, *popup*, etc., methods. When a dialog does not have an icon explicitly set, the default ooDialog icon is used. Note that none of the *standard* dialogs set an application icon. The *defaultIcon* method can be used to change the application icon from the default ooDialog icon to any icon the programmer wants. This is especially useful when a program puts up more than one dialog and / or includes a number of the standard dialogs. Setting the default icon can give all dialogs in a program a common look.

If the programmer wants to use a symbolic ID for the *id* argument, then the symbol must be in the global *.constdir*. This implies that the programmer needs to invoke the *defaultIcon* method, *after*, populating the **`.constDir`** with symbols.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example sets the default icon for the application using a stand alone icon file:

```
  .application~setDefaults('O', 'rc\oodStandardDialog.h')
  .application~defaultIcon('bmp\oodStandardDialog.ico')
```

31.2.9. parseIncludeFile

```
ResourceUtils::parseIncludeFile


>>--parseIncludeFile(--fileName--)---------------><
```

31.2.10. requiredOS

```
>>--requiredOS(--os--,--appName--)---------------><
```

Checks that an ooDialog program is executing on a required minimum Windows version.

Arguments:

The arguments are:

os

A key word specifying the minimum required version of Windows. The allowable key words are: *W2K, XP, W2K3, Vista,* and *Windows7*. Case is not significant.

appName [required]

This should be the name of the application that is executing. The name is used in the message box shown to the user.

Return value:

True if the current Windows OS mets the minimum requirement specified by *os*, otherwise false.

Remarks:

If the current OS does not meet the minimum version specified by *os*, a message box is shown to the user with the message stating: The *appName* application requires Windows *os* or later. It can not run on *current OS name*.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example comes from an ooDialog application that can not be used on XP. The very first line of the application checks that the current Windows version is at least Vista or later and ends the program if it is not. If the current OS is XP for instance, the *requiredOS* method will display a message to the user stating that: *The openSaveFileDemo.rex application requires Windows Vista or later. It can not run on XP.* before it returns:

```
  if \ .application~requiredOS('Vista', 'openSaveFileDemo.rex') then return 99
```

31.2.11. resolveNumericID

```
ResourceUtils::resolveNumericID


>>--resolveNumericID(--id--)---------------------><
```

31.2.12. resolveSymbolicID

```
ResourceUtils::resolveSymbolicID


>>--resolveSymbolicID(--id--)--------------------><
```

31.2.13. setDefaults

```
>>--setDefaults(-+-------+-+-----------+-+--------+-+---------+-+---------+-)--><
                 +-usage-+ +-,-symbols-+ +-,-auto-+ +-,-fName-+ +-,-fSize-+
```

The *setDefaults* method sets a number of default values for the application and / or performs tasks that effect the entire application, in a single method call.

Arguments:

The arguments are:

usage [optional]

The *usage* argument can either be a string specifying the global *.constdir strategy* to use for the application, or it can be a **`directory`** object specifying any number of defaults to set, and / or any number of application tasks to perform.

when *usage* is a string:

Specifies the usage *strategy* for the **`.constDir`**. *usage* must be exactly one of the following keywords. Only the first letter is required, and case is not significant:

Only	First	Last	Never

Only

Only the **`.constDir`** is used to resolve symbolic IDs.

First

When a symbolic ID needs to be resolved, the ooDialog framework first tries to resolve the ID through the **`.constDir`**. If that fails, the framework then tries the *constDir* attribute of the *dialog* object.

Last

This is the opposite of the *First* strategy. When a symbolic ID needs to be resolved, the ooDialog framework tries to resolve the ID through the *constDir* attribute of the *dialog* object. If that fails, the framework then tries the global **`.constDir`** object.

Never

The **`.constDir`** is never used to resolve symbolic IDs. This is the default when an ooDialog application starts up.

when *usage* is a directory

A **`directory`** object can be used to set some or all of the defaults and perform some or all of the tasks that the individual methods of the **`ApplicationManager`** do. The following indexes of the **`directory`** object are recognized. Indexes not in the list are ignored. For each index, the corresponding documentation for the method of the **`ApplicationManager`** may contain additional insight. E.g. restrictions on the value for the item at the index are identical to restrictions listed in the method documentation.

CONSTDIRUSAGE:

Specifies the **`.constDir`** usage *strategy*. This is equivalent to the *mode* argument in the *useGlobalConstDir* method and is the same as when *usage*is a string above.

SYMBOLSRC

Adds symbols to the **`.constDir`**. This is equivalent to the *symbolSrc* argument in the *useGlobalConstDir* method.

AUTODETECTION

Sets the default for *initAutoDetection* data detection. This is equivalent to the *on* argument in the *autoDetection* method.

FONTNAME

Sets the name of the default font for dialogs. This is equivalent to the *fontName* argument in the *defaultFont* method. With this exception, if both the FONTNAME and FONTSIZE indexes are missing from the **`directory`** object, the default is not set back to the default when an ooDialog application starts.

FONTSIZE

Sets the default font size for dialogs. This is equivalent to the *fontSize* argument in the *defaultFont* method. With this exception, if both the FONTNAME and FONTSIZE indexes are missing from the **`directory`** object, the default is not set back to the default when an ooDialog application starts.

symbols [optional]

symbols can either be the name of a file containing symbol *definitions*s, or a collection object that maps *symbolic* IDs to their numeric value.

This argument is ignored if a **`directory`** object is used for argument one.

auto [optional]

If *auto* is true, the default for *initAutoDetection* data detection is set to on, if false it is set to off.

This argument is ignored if a **`directory`** object is used for argument one.

fName [optional]

The new font, the name of the font. If omitted the global font name is not changed.

This argument is ignored if a **`directory`** object is used for argument one.

fSize [optional]

The new size for the default font. If omitted the global font size is not changed.

This argument is ignored if a **`directory`** object is used for argument one.

Return value:

True on success, false on error.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example specifies that only the **`.constDir`** should be used to resolve symbolic IDs, loads symbols into the **`.constDir`** from the file: **`rc\PropertySheetDemo.h`** and sets the default for automatic data detection to off:

```
.application~setDefaults("O", "rc\PropertySheetDemo.h", .false)

t1 = .ListViewDlg~new("res\PropertySheetDemo.dll", IDD_LISTVIEW_DLG)
t2 = .TreeViewDlg~new("res\PropertySheetDemo.dll", IDD_TREEVIEW_DLG)
t3 = .ProgressBarDlg~new("rc\PropertySheetDemo.rc", IDD_PROGRESSBAR_DLG)
t4 = .TrackBarDlg~new("res\PropertySheetDemo.dll", IDD_TRACKBAR_DLG)
t5 = .TabDlg~new("res\PropertySheetDemo.dll", IDD_TAB_DLG)

tabContent = .array~of(t1, t2, t3, t4, t5)
```

31.2.14. useGlobalConstDir

```
>>--useGlobalConstDir(--mode--+-------------+--)----------------><
                              +--symbolSrc--+
```

The ooDialog framework supports a *mechanism* for using *symbolic* resource IDs for method arguments that required a resource ID. The global *.constdir* is part of that support. The **`.constDir`** is the most efficient and capable way to use symbolic IDs. However, to support backwards program compatibility, the programmer must specify a usage *strategy* that includes the use of the **`.constDir`**.

By default the **`.constDir`** is not used. The programmer specifies the usage strategy for the **`.constDir`** through the *useGlobalConstDir* method. Optionally, the programmer can load symbols into the **`.constDir`** at the same time the **`.constDir`** usage is specified.

Arguments:

The arguments are:

mode [required]

Specifies the usage *strategy* for the **`.constDir`**. *mode*must be exactly one of the following keywords. Only the first letter is required, and case is not significant:

Only	First	Last	Never

Only

Only the **`.constDir`** is used to resolve symbolic IDs.

First

When a symbolic ID needs to be resolved, the ooDialog framework first tries to resolve the ID through the **`.constDir`**. If that fails, the framework then tries the *constDir* attribute of the *dialog* object.

Last

This is the opposite of the *First* strategy. When a symbolic ID needs to be resolved, the ooDialog framework tries to resolve the ID through the *constDir* attribute of the *dialog* object. If that fails, the framework then tries the global **`.constDir`** object.

Never

The **`.constDir`** is never used to resolve symbolic IDs. This is the default when an ooDialog application starts up.

symbolSrc [optional]

As a convenience the programmer can load symbols into the **`.constDir`** through the *symbolSrc* argument

symbolSrc can either be the name of a file containing symbol *definitions*s, or a collection object that maps *symbolic* IDs to their numeric value.

Return value:

True on success, false on failure.

Remarks:

If *symbolSrc* is a string it is taken to be the name of a file containing symbol definitons. The result in this case is the same as using the *parseIncludeFile*. The file is parsed and any symbol definitions are added to the **`.constDir`**.

When *symbolSrc* is not a string, it must be an object with a *supplier* method where the indexes the supplier provides are strings. The items the supplier provides are added to the **`.constDir`** using the supplied indexes.

Either the *addToConstDir* or the *parseIncludeFile* can be used to add additional symbols to the **.constDir**. **Note** that if the usage strategy is specified as *None* and the *symbolSrc* argument is used, the symbols will be loaded into the **.constDir**, (because the **.constDir** object is always present in an ooDialog.) However the symbols won't be available in the program, (because the programmer specified to not use the **.constDir**.)

Details

Raises syntax errors when incorrect arguments are detected.

Example:

In this example the programmer specifies to only use the **.constDir** to resolve symbolic IDs, and at the same time loads symbols into the **.constDir** from the **ticketWizard.h** file:

```
.application~useGlobalConstDir('O', "rc\ticketWizard.h")

msg = "The upcoming dialog demonstrates a Wizard 97 style dialog."
title = "Wizard 97 Dialog"
dlg = .TimedMessage~new(msg, title, 2000)
dlg~execute

...
```

31.3. .constDir object

The **.constDir** is a Rexx **.directory** object that is present in all ooDialog programs. It is initialized and placed in the **.local** environment automatically by the *.application*. The global **.constDir** is part of the *mechanism* allowing the use of *symbolic* IDs in ooDialog programs.

The **.constDir** is the most efficient way to use symbolic IDs. However, to maintain backwards compatibility with older ooDialog programs its use is turned off by default. The **.application** object, through its *useGlobalConstDir* method or *setDefaults* method is used to specify how the **.constDir** is used.

Other than specifying *how* the **.constDir** is to be used, the **.constDir** is mostly transparent to the programmer. To take best advantage of the **.constDir** the programmer should read the section in the Brief Overview chapter that discusses the symbolic ID *mechanism* in ooDialog.

This example shows a typical use of the **.constDir** in a program. The example sets the program to use **only** the **.constDir** and populates the **.constDir** with the symbolic IDs contained in the file: **accountingApp.h**. Notice that the **.constDir** is not even directly referenced.

```
.application~useGlobalConstDir('O', 'accountingApp.h')

dlg = .MainMenuDlg~new("QuickBooks.dll", IDD_MAIN_MENU_DIALOG)
if dlg~initCode == 0 then do
    dlg~execute("SHOWTOP", IDI_BOOKS)
end
...
```

Much of the time the programmer never needs to directly access the **.constDir**. However, the use of symbolic IDs can make a program more readable and / or maintainable. In this example, all the radion buttons are connected to one event handling method and which radio button was selected is determined using the **.constDir**:

```
::method onRbSelect
  expose process
  use arg info, handle

  id = .DlgUtil~loWord(info)

  select
    when id == .constDir[IDC_RB_ANYWHERE        ] then process = "ANYWHERE"
    when id == .constDir[IDC_RB_DIALOG          ] then process = "DIALOG"
    when id == .constDir[IDC_RB_ON_EDIT_OK      ] then process = "ON_EDIT_OK"
    when id == .constDir[IDC_RB_FOCUSED_ANYWHERE] then process = "FOCUSED_ANYWHERE"
    when id == .constDir[IDC_RB_FOCUSED_DIALOG  ] then process = "FOCUSED_DIALOG"
    when id == .constDir[IDC_RB_FOCUSED_EDIT    ] then process = "FOCUSED_EDIT"
    otherwise
      nop
  end
  -- End select
```

31.4. DlgUtil Class

All the methods of the DlgUtil class are class methods. There are no instance methods, other than inherited methods, for this class. The class methods are a collection of common utilities, mostly for converting different Windows values, getting version information, and the like.

31.4.1. Method Table

The **`DlgUtil`** class implements the methods listed in the following table.

Table 31.3. Methods of the DlgUtil class

Method	Description
Class Methods	**Class Methods**
and	*and's* two numbers and returns the result.
comCtl32Version	Determines the Common Controls Library version in use by ooDialog.
errMsg	Returns the error message text for the specified Windows system error code.
getGUID	Returns a GUID in string format.
getSystemMetrics	Obtains the system metric value for a given index.
hiWord	Returns the high-order word portion of a number as a number in the range of 0 through 65535.
loWord	Returns the low-order word portion of a number as a number in the range of 0 through 65535.
makeLParam	Packs two numbers into a LPARAM argument and returns the result.
makeWParam	Packs two numbers into a WPARAM argument and returns the result
or	*or's* two or more numbers and returns the result.
screenArea	Retrieves the usable screen area (work area.) on the primary display monitor.
screenSize	Retrieves the screen size in either pixels, dialog units, or both.
shiftLeft	Performs a logical bitwise shift left on a non-negative whole number.
shiftLeft	Performs a logical bitwise shift right on a non-negative whole number.
sHiWord	Returns the high-order word portion of a number as a number in the range of -32768 through 32767.
signed	Casts an unsigned whole number to its signed equivalent.
signed32	Casts a 32-bit unsigned whole number to its 32-bit signed equivalent.

Method	Description
sLoWord	Returns the low-order word portion of a number as a number in the range of -32768 through 32767.
sShiftLeft	Performs a logical bitwise shift left on a whole number.
sShiftLeft	Performs a logical bitwise shift right on a whole number.
unsigned	Casts a signed whole number to its unsigned equivalent.
unsigned32	Casts a 32-bit signed whole number to its 32-bit unsigned equivalent.
threadID	Returns the thread identifier of the thread the method is executing in.
version	Returns the ooDialog version string.
windowFromPoint	Retrieves the window handle of the window that contains the specified point.

31.4.2. and (Class Method)

```
>>--and(--number1-,-number2--)-------------------><
```

Combines two numbers into a single number by *anding* the individual numbers together. This is the typical *and* operation used by assembly and C programmer's. The typical ooDialog programmer would have no need of the method. It is provided as a convenience for use in sophisticated ooDialog programs.

The normal use of an *and* operation in ooDialog programs is to extract the low or high words from an argument returned from the operating system. The addition of the *loWord*() and *hiWord*() methods to the ooDialog framework have eliminated most of the reason for this usage.

Arguments:

The required two arguments are the whole numbers to be *anded* together.

As a convenience, the whole numbers can also be specified as a string with the following format: **`0x`** followed by a series of hexadecimal digits with no spaces. Such as **`"0xff12abc9"`** or **`"0x1"`** or **`"0x00000080"`**.

Return value:

The return value is the number that is the result of *anding* the two arguments together.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

The following example is part of a generic method in an application that returns the text of an edit control. The application is a complex one and has many edit controls that are read only. The text for read only controls is ignored. The method returns .nil for read only controls to differentiate the difference between a read only control and a control that has the empty string.

The value of the ES_READONLY style is 0x0800.

```
::method getEditText private
  use strict arg control

  -- If the control is read only, we just return .nil
  if .DlgUtil~and(control~getStyleRaw, 0x0800) <> 0 then return .nil
  ...
```

31.4.3. comCtl32Version (Class Method)

```
>>--comCtl32Version(--+----------+-)-------------><
                      +--format--+
```

Use this method to determine the version of the Common Controls *Library* used by ooDialog on the current system.

Arguments:

The single argument is:

format [optional]

A keyword indicating the format of the returned version string. Only the first letter is needed and case is not significant. Any unrecognized keyword is ignored and the default format is used. The format for each keyword is as follows:

Short

The version number, for instance **`4.72`** or **`6.0`**. This is the default.

Number

This is an alias for short.

OS

The minimum operating system that can be expected to be compatible with the library. Earlier versions of the common control library were also distributed with Internet Explorer and so an OS part of say, **`W98 / IE 4.01`** would indicate that the current common control library supports all the features available on Windows 98 or with Internet Explorer 4.01.

Full

The full format string in the format **`comctl32.dll version 6.0 (XP)`**. Where the 6.0 is the number part and XP is the OS part.

Return value:

This method returns a string in one of the formats described above.

Example:

In the following example, the programmer repositions the dialog controls under certain conditions. In Windows XP and later, this task is more accurate if the **`getIdealSize()`** method of the Button class is used. However, the programmer needs the application to also run on Windows 2000. He uses the **`comCtl32Version()`** method to determine if the *getIdealSize* method is available. If it is not available he uses an alternative, but less accurate method to reposition the controls.

```
::method repositionControls

  if .DlgUtil~comCtl32Version < 6.0 then return self~doW2KReposition

  size = self~newPushButton(IDC_CHECK_TWO)~getIdealSize
  if size == .nil then return .false

  -- do the repositioning

return .true
```

An alternative way to achieve the same thing might be:

```
::method repositionControls

  if .DlgUtil~comCtl32Version('O') == "XP" then return self~doXPReposition

  -- Do the less accurate repositioning
  ...

return .true
```

31.4.4. errMsg

```
>>--errMsg(--errCode--)--------------------------><
```

Returns the error message text for the specified Windows system error code.

Arguments:

The single argument is:

errCode [required]

The Windows system error code to get the error message for. Typically this would come from the *.SystemErrorCode* variable, but it can be any non-negative whole number in the range 0 to 4294967295.

For convenience in looking up COM or OLE related error codes, strings in conventional *hexadecimal* format are also accepted. Typically an **`OleObject`** COM related error is displayed something like: 0x80040000.

Return value:

Returns the error message text that Windows associates with the specified error code. Note that the error message comes from Windows, not ooDialog. ooDialog has no control over the clarity of the message.

Remarks:

Although any valid whole number within the range specified is accepted, only numbers that are actually valid system errors will produce a valid result. Numbers that are not Windows system errors will return a string similar to, *Internal Windows error formatting the message (317)*. The number in parenthesis will be the system error code that Windows returned when asked for the message text for the *errCode* argument. Since 317 is a valid system error, its message text could be looked up using the *errMsg* method, see the example below.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows several usages of the *errMsg* method:

```
  say .DlgUtil~errMsg('0x80004024')
  say .DlgUtil~errMsg(2147500068)
  say
  say .DlgUtil~errMsg(266)
  say
  say .DlgUtil~errMsg(279)
  say
  say .DlgUtil~errMsg(317)
  say
  say .DlgUtil~errMsg(87)

/* Output would be:
Error code 2147500068 (0x80004024): The specified activation could not occur in the
 client context as specified.
Error code 2147500068 (0x80004024): The specified activation could not occur in the
 client context as specified.

Error code 266 (0x0000010a): The copy functions cannot be used.

Internal Windows error formatting the message (317)

Error code 317 (0x0000013d): The system cannot find message text for message number 0x%1
 in the message file for %2.

Error code 87 (0x00000057): The parameter is incorrect.

*/
```

31.4.5. getGUID

```
>>--getGUID(--+----------------+--)--------------><
              +--conventional--+
```

Returns a GUID in string format.

Arguments:

The single argument is:

conventional [optional]
: Specifies the format of the returned GUID. If true the returned string format is Microsoft's conventional format. If false the string is returned in universal format. If omitted, *conventional* is false.

Return value:

A GUID in the string format specified, or the **`.nil`** object on error.

Remarks:

A GUID is actually a number, a 128-bit number. That large a number is difficult to display and the conventional string format is commonly used when talking about a GUID. It is even difficult to pass that large a number as an argument to a function or method. Often software accepts the string format of a GUID and internally converts it to a number.

A new GUID is generated for each invocation of this method.

By default the string GUID will be similar to: **`3d2c9438-a3b0-494d-ba5d-10f53e6ec9cf`**

If Microsoft's convention is requested the same GUID would be returned as: **`{3d2c9438-a3b0-494d-ba5d-10f53e6ec9cf}`**

A GUID and a UUID are synonymous and can be used interchangeably in ooDialog.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example prints out a new GUID every time it is executed. Typically, the GUID would then be copy-pasted into the software it will be used in:

```
/* genGUID.rex - simple GUID generating program. */

    guid = .DlgUtil~getGUID
    say guid
return 0
::requires 'ooDialog.cls'
```

31.4.6. getSystemMetrics (Class Method)

```
>>--getSystemMetrics(--index--)------------------><
```

Obtains the system metric value for the given index.

Arguments:

The only argument is:

index

The numeric index of the system metric.

Return value:

The return value is dependent on the index queried. See the *remarks* section.

Remarks:

Good documentation on the system metrics is found in the MSDN *documentation* under the GetSystemMetrics function. This documentation contains both the numeric value of the different indexes and information on what is returned for each index. The documentation is easy to found using a Google search of "GetSystemMetrics MSDN library"

The OS will return 0 if an invalid index is used. However, the return value for some indexes is also 0. The programmer will need to determine from the context if 0 is an error return or not.

Example:

The following code snippet is from an application where the user can have 5, 6, or more, independent dialogs open at one time. One of the menu options is "Tile Dialogs." When the user selects this option all the open dialogs are "tiled."

All the open dialog objects are stored in a queue. In the **onTile** method, which is invoked when the user selects the menu item, each dialog is fetched in turn from the queue. Then the dialog is repositioned at an offset from the preceding dialog. It is shifted to the right the width of 2 scroll bars and shifted down the width of the title bar. (This width is the title bar width plus the thin border around the title bar.) The height of the thin border, the height of the title bar, and the width of a scroll bar are all determined by querying the system metrics.

```
::method onTile
  expose offSetX offSetY dialogQueue

  -- SM_CXVSCROLL = 20
  -- SM_CYCAPTION = 4
  -- SM_CYBORDER  = 6

  if \ offSetX~datatype('W') then do
    scrollBarX = .DlgUtil~getSystemMetrics(20)
    titleBarY = .DlgUtil~getSystemMetrics(4)
    windowBorderY = .DlgUtil~getSystemMetrics(6)

    offSetX = 2 * scrollBarX
    offSetY = (2 * windowBorderY) + titleBarY
  end

  parse value self~getWindowRect(self~dlgHandle) with x y .

  do dlg over dialogQueue
    x += offSetX
    y += offSetY

    self~setWindowRect(dlg~dlgHandle, x, y, 0, 0, "NOSIZE")
  end
```

31.4.7. hiWord (Class Method)

```
>>--hiWord(--param--)----------------------------><
```

The Windows API often packs two values into a single number. One value in the low-order word and the other value in the high-order word. In ooDialog this is most often seen in the arguments passed to event notification methods. This utility method is used to extract the high-order word from a number.

Arguments:

The arguments are:

param [required]

The number whose high-order word is needed.

Return value:

The return is the high-order word portion of the number. This whole number value will be in the range of 0 through 65535.

Remarks:

Unfortunately, the practice of packing two values into a single number can lead to ambiguities, especially when the values can be either negative or positive, and start getting larger. The number returned from the *hiWord* method will always be non-negative. For situations where the expected returned number may be negative, the *sHiWord* method is provided.

Example:

See the loWord() *example*.

31.4.8. loWord (Class Method)

```
>>--loWord(--param--)----------------------------><
```

The Windows API often packs two values into a single number. One value in the low-order word and the other value in the high-order word. In ooDialog this is most often seen in the arguments passed to event notification methods. This utility method is used to extract the low-order word from a number.

Arguments:

The arguments are:

param [required]

The number whose low-order word is needed.

Return value:

The return is the low-order word portion of the number. This whole number value will be in the range of 0 through 65535.

Remarks:

Unfortunately, the practice of packing two values into a single number can lead to ambiguities, especially when the values can be either negative or positive, and start getting larger. The number returned from the *loWord* method will always be non-negative. For situations where the expected returned number may be negative, the *sLoWord* method is provided.

Example:

A common place in the Windows API where two values are packed into a single number is where the API deals with position. This example shows how to extract the x and y position coordinates of a dialog after it has been moved. The example is complete, it can be cut and pasted into a file and will execute as is.

```
  dlg = .MovingDialog~new
  dlg~createCenter(100, 60, "Move Me")
  dlg~execute("SHOWTOP")

::requires 'ooDialog.cls'

::class 'MovingDialog' subclass UserDialog

::method init
  forward class (super) continue

  self~connectMove(onMove)

::method onMove
  use arg unUsed, posInfo

  x = .DlgUtil~loWord(posInfo)
  y = .DlgUtil~hiWord(posInfo)
  say 'At coordinate (' x',' y' ) on the screen. (In pixels.)'
```

31.4.9. makeLParam (Class Method)

```
>>--makeLParam(--numLow--,--numHigh--)----------><
```

When a *message* is *sent* to an *underlying* window two of the arguments are the WPARAM and the LPARAM parameters. Quite often the Windows API requires that two numbers be packed into the LPARAM argument. The *makeLParam* method is a convenience method used to perform the packing. It is similar to the *loWord* and *hiWord* methods, but works in the opposite direction.

Arguments:

The arguments are:

numLow [required]

A whole number to be packed into the low word position of the LPARAM number.

numHigh [required]

A whole number to be packed into the high word position of the LPARAM number.

Return value:

The return is a number suitable for the LPARAM argument in a message sent to an underlying window, when the message requires two values in LPARAM.

Details

Raises syntax errors when incorrect arguments are detected.

Example

This example shows the implementation of a method used to set the margins in an edit control

```
::method setEditMargins
    use arg left, right

    EC_LEFTMARGIN  = "0x0001"
    EC_RIGHTMARGIN = "0x0002"
    EC_USEFONTINFO = "0xFFFF"
    EM_SETMARGINS  = "0x00D3"

    if arg(1, 'O'), arg(2, 'O') then do
       self~sendMessage(EM_SETMARGINS, 0, EC_USEFONTINFO)
    end
    else do
       flag = 0
       if arg(1, 'E') then flag = EC_LEFTMARGIN
       if arg(2, 'E') then flag = .DlgUtil~or(flag, EC_RIGHTMARGIN)
       lParam = .DlgUtil~makeLParam(left, right)
       self~sendMessage(EM_SETMARGINS, flag, lParam)
    end
    return 0
```

31.4.10. makeWParam (Class Method)

```
>>--makeWParam(--numLow--,--numHigh--)-----------><
```

When a *message* is *sent* to an *underlying* window two of the arguments are the WPARAM and the LPARAM parameters. Quite often the Windows API requires that two numbers be packed into the WPARAM argument. The *makeWParam* method is a convenience method used to perform the packing. It is similar to the *loWord* and *hiWord* methods, but works in the opposite direction.

Arguments:

The arguments are:

numLow [required]

A whole number to be packed into the low word position of the WPARAM number.

numHigh [required]

A whole number to be packed into the high word position of the WPARAM number.

Return value:

The return is a number suitable for the WPARAM argument in a message sent to an underlying window, when the message requires two values in WPARAM.

Details

Raises syntax errors when incorrect arguments are detected.

Example

Although the *example* for *makeLParam* does not show the use of *makeWParam*, the basic idea in using either method is essentially the same.

31.4.11. or (Class Method)

```
          +-,-------+
          V         |
>>--or(----number--+--)--------------------------><
```

Combines any number of numerical values into a single number by *oring* the individual numbers together. This is the typical or operation used by assembly and C programmer's. The ooDialog programmer does not need to understand the concept to use this method. Some arguments to some methods in the ooDialog framework can take a value that is a combination of some individual *flags*. The or method is provided to construct this combination.

Arguments:

There is no restriction on the number of arguments to this method, other than each argument must represent a whole number.

As a convenience, the whole numbers can also be specified as a string with the following format: **`0x`** followed by a series of hexadecimal digits with no spaces. Such as **`"0xff12abc9"`** or **`"0x1"`** or **`"0x00000080"`**. This makes it easy to look up and use the value of a flag directly. For instance looking up the value of **`ILC_COLOR24`**, one would see it is **`0x00000018`**. This value could be used directly in the **`or()`** method. Case is not significant in the string.

Return value:

The return value is a number that is the result of combining all the individual arguments together.

Details:

Raises syntax errors when incorrect arguments are detected.

Example:

One method that has an argument that can be the result of combining a number of flags is the *create* method of the *Image* class. This is how the or method might be used

```

  flags = .DlgUtil~or(.Image~toID(ILC_COLOR24), .Image~toID(ILC_MASK))

  -- Create an empty .ImageList object:
  imageList = .ImageList~create(.Size~new(61, 46), flags, 10, 10);
```

The following is equivalent to the above example:

```
  flags = .DlgUtil~or(0x00000018, 0x00000001)

  -- Create an empty .ImageList object:
  imageList = .ImageList~create(.Size~new(61, 46), flags, 10, 10);
```

To round out the examples a little:

```
  flags = .DlgUtil~or(.Image~toID(ILC_COLOR24), .Image~toID(ILC_MASK))
  say "Flags value:" flags

  flags = .DlgUtil~or(0x00000018, 0x00000001)
  say "Flags value:" flags

::requires 'ooDialog.cls'

/* Output would be:
     Flags value: 25
     Flags value: 25
*/
```

31.4.12. screenArea (Class Method)

```
>>--screenArea-----------------------------------><
```

Gets the usable screen area (work area.) on the primary display monitor. The work area is the portion of the screen not obscured by the system taskbar or by application desktop toolbars.

Arguments:

This method takes no arguments.

Return value:

A *Rect* object that contains the coordinates of the work area. The coordinates are expressed in screen *coordinates*.

31.4.13. screenSize (Class Method)

```
>>--screenSize(--+--------+--+-----------+--)----><
                 +--flag--+  +-,-dlgObj--+
```

Retrieves the screen size in either pixels, dialog units, or both.

Arguments:

The arguments are:

flag [optional]

A keyword signaling whether to return the size in pixels, dialog units, or both. The default is both. The keywords are PIXELS, DIALOGUNITS, and BOTH. Only the first letter is needed and case is insignificant.

dlgObj [optional]

A Rexx dialog object. When this argument is used, dialog units are calculated correctly for the dialog. If it is omitted, the dialog units will most likely be incorrect. See the remarks section

Return value:

The return is dependent on the flag in use:

Pixels:

A *Size* object with the screen size in pixels.

DialogUnits

A *Size* object with the screen size in dialog units.

Both

An **`Array`**> with the screen size in dialog units at indexes 1 and 2 and the size in pixels at indexes 3 and 4. Indexes 1 and 3 contain the width, indexes 2 and 4 contain the height..

Remarks:

If a Rexx dialog object is not supplied, it is unlikely that the *dialog unit*s will be correct. Without a reference to a dialog, the method calculates the dialog units using 8 point System font. Since dialog units are dependent on the font in use for a specific dialog, It is highly unlikely that this is correct for any specific dialog.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example calculates the mid-point of the upper left quadrant of the screen and positions the dialog at that point:

```
::method initDialog

  -- Get the screen size and then position ourself in the upper left quadrant.
  s = .DlgUtil~screenSize('P')
  p = .Point~new(s~width % 4, s~height % 4)
  self~moveTo(p, "SHOWWINDOW")
```

31.4.14. shiftLeft

```
>>--shiftLeft(--unsignedN--+----------+--)------->< 
                           +-,-count--+
```

Performs a logical bitwise shift left.

Arguments:

The arguments are:

unsignedN

A non-negative whole number up to 64 bits in size. That is, 0 to 18,446,744,073,709,551,615.

count

The non-negative whole number of bits to shift. *count* must be in the range of 0 to 63.

Return value:

The unsigned whole number that is the result of the logical shift.

Details

Raises syntax errors when incorrect arguments are detected.

31.4.15. shiftRight

```
>>--shiftRight(--unsignedN--+----------+--)------>< 
                            +-,-count--+
```

Performs a logical bitwise shift right.

Arguments:

The arguments are:

unsignedN

A non-negative whole number up to 64 bits in size. That is, 0 to 18,446,744,073,709,551,615.

count

The non-negative whole number of bits to shift. *count* must be in the range of 0 to 63.

Return value:

The unsigned whole number that is the result of the logical shift.

Details

Raises syntax errors when incorrect arguments are detected.

31.4.16. sHiWord (Class Method)

```
>>--sHiWord(--param--)---------------------------><
```

The Windows API often packs two values into a single number. One value in the low-order word and the other value in the high-order word. The *sHiWord* method is the signed version of the *hiWord* method. It is used to extract the high-order word from a number, where the value of the high-order word may be positive or negative.

Arguments:

The arguments are:

param [required]

The number whose high-order word is needed.

Return value:

The return is the high-order word portion of the number. The returned whole number will be in the range of -32768 to 32767.

Remarks:

Unfortunately, the practice of packing two values into a single number can lead to ambiguities, especially when the values can be either negative or positive, and start getting larger. The number returned from the *sHiWord* method can be either negative or positive, but the largest maximum number will only be 32767. For situations where the expected returned number is always positive and may be larger than 32767, the *hiWord* method is provided.

Example:

See the *sLoWord* *example*.

31.4.17. signed

```
>>--signed(--unsigned--)-------------------------><
```

Casts an unsigned whole number to its signed equivalent.

Arguments:

The single argument is:

unsigned

The unsigned whole number to cast.

Return value:

The signed equivalent of *unsigned*.

Remarks:

The *signed* method works with whole numbers of the same size as the addressing mode of the interpreter. I.e., if the address mode is 32-bit, *signed* works with 32-bit numbers and if the addressing mode is 64-bit it works with 64-bit numbers.

In 32-bit addressing mode, *signed* accepts a whole number in the range of 0 to 4,294,967,295 and returns a whole number in the range of -2,147,483,648 to 2,147,483,647.

In 64-bit addressing mode, *signed* accepts a whole number in the range of 0 to 18,446,744,073,709,551,615 and returns a whole number in the range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

31.4.18. signed32

```
>>--signed(--unsigned--)-------------------------><
```

Casts a 32-bit unsigned whole number to its 32-bit signed equivalent.

Arguments:

The single argument is:

unsigned

The unsigned 32-bit whole number to cast.

Return value:

The 32-bit signed equivalent of *unsigned*.

Remarks:

The *signed* method works with whole numbers of the same size as the addressing mode of the interpreter. I.e., if the address mode is 32-bit, *signed* works with 32-bit numbers and if the addressing mode is 64-bit it works with 64-bit numbers.

In 32-bit addressing mode, *signed* accepts a whole number in the range of 0 to 4,294,967,295 and returns a whole number in the range of -2,147,483,648 to 2,147,483,647.

In 64-bit addressing mode, *signed* accepts a whole number in the range of 0 to 18,446,744,073,709,551,615 and returns a whole number in the range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

In general, the *signed* and *unsigned* methods work well. However, there is a problem in 64-bit addressing mode when trying to convert a number known to be 32-bit. A number such as 0xfffffdd8 is positive 4294966744 if treated as a 64-bit number. However it is -552 if treated as a 32-bit number. The *signed32* and *unsigned32* methods are for use in these cases.

31.4.19. sLoWord (Class Method)

```
>>--sLoWord(--param--)---------------------------><
```

The Windows API often packs two values into a single number. One value in the low-order word and the other value in the high-order word. The *sLoWord* method is the signed version of the *loWord* method. It is used to extract the low-order word from a number, where the value of the low-order word may be positive or negative.

Arguments:

The arguments are:

param [required]

The number whose low-order word is needed.

Return value:

The return is the low-order word portion of the number. The returned whole number will be in the range of -32768 to 32767

Remarks:

Unfortunately, the practice of packing two values into a single number can lead to ambiguities, especially when the values can be either negative or positive, and start getting larger. The number returned from the *sLoWord* method can be either negative or positive, but the largest maximum number will only be 32767. For situations where the expected returned number is always positive and may be larger than 32767, the *loWord* method is provided.

Example:

One place where two values are packed into one number by the operating system is the return from scrolling an edit control. The operating system returns true or false in the high word, for success or failure. The lines scrolled are in the low word. When the edit control is scrolled from the bottom towards the top, the number of lines is negative. This example shows the result of using the *scrollCommand* repeatedly until the top of the edit control is reached:

```
  success = .true
  do while success
    ret = editControl~scrollCommand("PAGEUP")

    success = .DlgUtil~sHiWord(ret)
    linesMoved = .DlgUtil~sLoWord(ret)

    say "Scrolled the edit control:"
    say "  Success?" success
    say "  Lines:   " linesMoved
  end

/* Output might be:

Scrolled the edit control:
  Success? 1
  Lines:   -16
Scrolled the edit control:
  Success? 1
  Lines:   -16
...
Scrolled the edit control:
  Success? 1
  Lines:   -16
Scrolled the edit control:
  Success? 1
  Lines:   -5
Scrolled the edit control:
  Success? 0
  Lines:   0

*/
```

31.4.20. sShiftLeft

```
>>--sShiftLeft(--signedN--+----------+--)-------><
                          +-,-count--+
```

Performs a logical bitwise shift left on a whole number.

Arguments:

The arguments are:

signedN

A whole number up to 64 bits in size. That is, -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

count

The non-negative whole number of bits to shift. *count* must be in the range of 0 to 63.

Return value:

The whole number that is the result of performing the logical shift.

Details

Raises syntax errors when incorrect arguments are detected.

31.4.21. sShiftRight

```
>>--sShiftRight(--signedN--+----------+--)-------><
                           +-,-count--+
```

Performs a logical bitwise shift right on a whole number.

Arguments:

The arguments are:

signedN

A whole number up to 64 bits in size. That is, -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

count

The non-negative whole number of bits to shift. *count* must be in the range of 0 to 63.

Return value:

The whole number that is the result of performing the logical shift.

Details

Raises syntax errors when incorrect arguments are detected.

31.4.22. threadID (Class Method)

```
>>--threadID-------------------------------------><
```

Returns the thread identifier of the thread the method is executing in.

Arguments:

There are no arguments for this method.

Return value:

The return is a non-zero positive number, the number the operating system uses to identify the current thread.

Remarks:

Until the thread terminates, the thread identifier uniquely identifies the thread throughout the operating system.

This is a convenience method for the use of the developers more than anything. However, for any user of ooRexx it does provide some insight into the concurrency functionality of ooRexx.

Example:

This example displays the thread ID from several different threads, (activities,) during the life cycle of a dialog.

```
  dlg = .SimpleDialog~new
  say "In the start up of the program thread ID:  " .DlgUtil~threadID
  dlg~execute("SHOWTOP", IDI_DLG_OOREXX)

::requires "ooDialog.cls"

::class 'SimpleDialog' subclass UserDialog

::method init
  forward class (super) continue

  self~create(30, 30, 257, 123, "Simple Dialog", "CENTER")

::method defineDialog

  self~createPushButton(IDOK, 142, 99, 50, 14, "DEFAULT", "Ok")
  self~createPushButton(IDCANCEL, 197, 99, 50, 14, , "Cancel")

::method initDialog

  say "In initDialog() thread ID:                 " .DlgUtil~threadID
  self~start("secondActivity")

::method secondActivity unguarded

  reply 0
  j = SysSleep(1)
  say "Executing in the second activity thread ID:" .DlgUtil~threadID

::method ok unguarded

  say "In the ok method thread ID:                " .DlgUtil~threadID
  return self~ok:super

/* The output on the console might ge:

In the start up of the program thread ID:   128
In initDialog() thread ID:                  128
Executing in the second activity thread ID: 2672
In the ok method thread ID:                 2344

*/
```

31.4.23. unsigned

```
>>--unsigned(--signed--)-------------------------><
```

Casts a signed whole number to its unsigned equivalent.

Arguments:

The single argument is:

signed

The signed whole number to be cast.

Return value:

The unsigned whole number equivalent to *signed*.

Remarks:

The *unsigned* method works with whole numbers of the same size as the addressing mode of the interpreter. I.e., if the address mode is 32-bit, *unsigned* works with 32-bit numbers and if the addressing mode is 64-bit it works with 64-bit numbers.

In 32-bit addressing mode, *unsigned* accepts a whole number in the range of -2,147,483,648 to 2,147,483,647 and returns a whole number in the range of 0 to 4,294,967,295.

In 64-bit addressing mode, *unsigned* accepts a whole number in the range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 and returns a whole number in the range of 0 to 18,446,744,073,709,551,615.

31.4.24. unsigned32

```
>>--unsigned32(--signed--)-----------------------><
```

Casts a 32-bit signed whole number to its 32 bit-unsigned equivalent.

Arguments:

The single argument is:

signed

The signed 32-bit whole number to be cast.

Return value:

The unsigned 32-bit whole number equivalent to *signed*.

Remarks:

The *signed* method works with whole numbers of the same size as the addressing mode of the interpreter. I.e., if the address mode is 32-bit, *unsigned* works with 32-bit numbers and if the addressing mode is 64-bit it works with 64-bit numbers.

In 32-bit addressing mode, *unsigned* accepts a whole number in the range of -2,147,483,648 to 2,147,483,647 and returns a whole number in the range of 0 to 4,294,967,295.

In 64-bit addressing mode, *unsigned* accepts a whole number in the range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 and returns a whole number in the range of 0 to 18,446,744,073,709,551,615.

In general, the *signed* and *unsigned* methods work well. However, there is a problem in 64-bit addressing mode when trying to convert a number known to be 32-bit. A number such as 0xffffdd8 is positive 4294966744 if treated as a 64-bit number. However it is -552 if treated as a 32-bit number. The *unsigned32* and *signed32* methods are for use in these cases.

31.4.25. version (Class Method)

```
>>-version(--+---------+-)-----------------------><
             +-format--+
```

Returns the ooDialog version string.

Arguments:

The single argument is:

format [optional]

A keyword indicating the format of the returned version string. Only the first letter is needed and case is not significant. Any unrecognized keyword is ignored and the default format is used. The format for each keyword is as follows:

Full

A string specifying the version of ooDialog. This is the default. The string has the format: **`ooDialog Version x.x.x.SS (an ooRexx Windows Extension)`** where x.x.x is the ooDialog version and SS is the Subversion revision number of the build. The actual version string might look like: **`ooDialog Version 4.2.0.8038 (an ooRexx Windows Extension)`**

Short

Only the x.x.x.SS part of the full version string is returned. Using the above example, the string would look like **`4.2.0.8038`** .

Level

The ooDialog *level*. Continuing with the same example the level would be **`4.2.0`**

Return value:

A string in one of the formats specified above.

Example:

```
v = .DlgUtil~version
say v
say

if .DlgUtil~version("SHORT") < 4.1.0.0000 then do
  say 'Too bad, you are still running ooRexx 4.0.0'
end
else do
  if .DlgUtil~version("LEVEL") < 4.2.0 then
    say 'You should really get a 4.2.0 ooDialog, it is much improved.'
end

::requires 'ooDialog.cls'

/* On the screen you might see:

ooDialog Version 4.0.0.3768 (an ooRexx Windows Extension)

Too bad, you are still running ooRexx 4.0.0

*/
```

31.4.26. windowFromPoint

```
>>--windowFromPoint(--p--)-----------------------><
```

Retrieves the window *handle* of the window that contains the specified point.

Arguments:

The single argument is:

p [required]

A *Point* object that specifies the point to be tested.

Return value:

Returns a handle to the window that contains the point. If no window exists at the point then 0 is returned. For a *Static* text control, the handle to the window under the control is returned.

Remarks:

This method does not retrieve handles of hidden or disabled windows even if the point is within the window. The *childWindowFromPoint* method of the *dialog* object can be used for a nonrestrictive search.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows a method that tests if the mouse is over a list view control and that there is no other window covering the list view at that point:

```
::method overListView private

  if \ self~isDialogActive then return .false

  mouse = .Mouse~new(self)
  pos = mouse~getCursorPos

  hwndLV = self~newListView(IDC_LV)~hwnd

  if .DlgUtil~windowFromPoint(pos) == hwndLV then return .true
  else return .false
```

31.5. FindWindow Routine

Searches the Windows application list for a specific window and returns its handle.

```
>>-FindWindow(--Caption--)-----------------------><
```

Argument:

The only argument is:

Caption

Caption of the window that is to be searched.

Return value:

Handle of the window or 0 if the window was not found.

31.6. Locate

```
>>--Locate(--+----------+--)---------------------><
             +--update--+
```

Returns the directory containing the Rexx source code file that *Locate* is executing in. This is similar to *parse source*, but returns the complete directory path rather than the complete file name. In addition, *Locate* sets the value of the *srcDir* attribute of the *.application* to the complete directory path name.

Arguments:

The single argument is:

update [optional]

By default this routine only sets the **.application's** *srcDir* attribute the first time it is invoked. On successive invocations, it does not change the attribute. If update is true, it resets the **.application's** *srcDir* attribute to the value it is returning.

Return value:

Returns the complete path name, including the trailing slash (\,) of the directory the source code file containing the executing *Locate* is located in.

Remarks:

The primary purpose of the *Locate* function is to help the ooDialog programmer create complete path names for any resource files, or other auxiliary files, needed by an application. This in turns allows the application to be executed from any directory. The *Locate* function is a convenience function. The same functionality can be achieved using *parse source* and some mechanism for retaining the directory value.

Typically, *Locate* function would be executed once during the start up of an application and the *srcDir* attribute of the *.application* object would be used elsewhere in the application to gain access to the directory value. If the programmer chooses to execute *Locate* more than once, then she would need to decide if the *srcDir* attribute's value should be updated with the new directory value. Assuming, *Locate* is returning a new value. This is the purpose of the *update* argument. By default, the *srcDir* attribute will not be changed. Pass true as the first argument to *Locate* to force the routine to update the *srcDir* attribute.

Example:

This example comes from an application that has its resource files located in a subdirectory of the main application directory. The *Locate* function is used to get the complete path name of the directory the main program file is in and then the returned value is used to construct complete path names to the application's resource files:

```
  srcDir = Locate()

  rcFile = srcDir"resources\imageButton.rc"
  symbolFile = srcDir"resources\imageButton.h"

  .application~setDefaults("O", symbolFile, .false)

  dlg = .ImageListDlg~new(rcFile, IDD_IMAGELIST_BUTTON)
  ...

::class 'ImageListDlg' subclass RcDialog
  ...

::method setPictureButtons private
  expose pbView pbAdd stStatus imagesLoaded imageList
```

```
    if .DlgUtil~comCtl32Version < 6 then return self~oldSetButtons

    pbView~style = "MULTILINE BOTTOM"

    srcDir = .application~srcDir

    -- The images are loaded from files.
    files = .array~new()
    files[1] = srcDir"resources\Normal.bmp"       -- Normal
    files[2] = srcDir"resources\Hot.bmp"          -- Hot (hover)
    files[3] = srcDir"resources\Pushed.bmp"       -- Pushed
    files[4] = srcDir"resources\Disabled.bmp"     -- Disabled
    files[5] = srcDir"resources\Default.bmp"      -- Default button
    files[6] = srcDir"resources\Hot.bmp"          -- Stylus hot, tablet PC only
    ...
```

31.7. MSSleep Routine

Sleeps for the specified number of milliseconds.

```
>>-MSSleep(--duration--)-------------------------><
```

Argument:

The only argument is:

duration

The time in milliseconds to sleep.

Return value:

This function always returns 0.

31.8. OS Class

The **OS** class provides methods for interacting with the operating system a program is currently executing on. Many of the methods extract information from or about the operating system. Other methods set operating system wide information or notify the operating system of changes.

All the methods of the OS class are class methods. In general these methods are easy to use, returning true or false about a specific piece of information. See the *isAtLeastVista*() method for an example of usage.

31.8.1. Method Table

The following table lists the class methods of the **OS** class:

Table 31.4. OS Class Method Reference

Method	Description
Class Methods	**Class Methods**
is32on64bit	Tests if running under ooRexx 32-bit on a 64 bit system
is64bit	Tests if running under ooRexx 64-bit
isAtLeastVista	Tests if operating system is Windows Vista or later

Method	Description
isAtLeastW2K	Tests if operating system is Windows 2000 or later
isAtLeastW2K3	Tests if operating system is Windows 2003 or later
isAtLeastWindows7	Tests if operating system is Windows 7 or later
isAtLeastXP	Tests if operating system is Windows XP or later
isServer2008	Tests if operating system is Windows Server 2008
isServer2008R2	Tests if operating system is Windows Server 2008 Release 2
isVista	Tests if operating system is Windows Vista
isW2K	Tests if operating system is Windows 2000
isW2K3	Tests if operating system is Windows 2003
isWindows7	Tests if operating system is Windows 7
isWow64	Alias for the *is32on64bit*() method
isXP	Tests if operating system is Windows XP
isXP32	Tests if operating system is 32-bit Windows XP
isXP64	Tests if operating system is 64-bit Windows XP
settingChanged	Notifies the operating system that environment settings have changed.
shellChangeNotify	Notifies the operating system that the application has performed an action that may affect the Shell.

31.8.2. is32on64bit (Class Method)

```
>>--is32on64bit----------------------------------><
```

Returns true if the currently executing program is running under the 32-bit ooRexx interpreter on a 64-bit version of Windows, otherwise false.

31.8.3. is64bit (Class Method)

```
>>--is64bit--------------------------------------><
```

Returns true if the currently executing program is running under the 64-bit ooRexx interpreter on a 64-bit version of Windows, otherwise false.

31.8.4. isAtLeastVista (Class Method)

```
>>--isAtLeastVista-------------------------------><
```

Returns true if the currently executing program is running on Windows Vista, or later. Otherwise it returns false.

Example:

The *MonthCalendar* class has a number of methods that are only available on Windows Vista or Windows 7. This example is from a program that demonstrates those methods. It first checks if the program is running on Vista, and if not it quits.

```
/* Simple MonthCalendar example */

  -- We demonstrate some of the MonthCalendar features here that are
  -- only available on Vista.  So, be sure we are running on Vista and
  -- abort if not.
  if \ .OS~isAtLeastVista then do
    info = "This program requires Windows Vista or later."
    return messageDialog(info, , "Error", "OK", "HAND")
  end
```

31.8.5. isAtLeastW2K (Class Method)

```
>>--isAtLeastW2K---------------------------------><
```

Returns true if the currently executing program is running on Windows 2000, or later. Otherwise it returns false.

31.8.6. isAtLeastW2K3 (Class Method)

```
>>--isAtLeastW2K3--------------------------------><
```

Returns true if the currently executing program is running on Windows 2003, or later. Otherwise it returns false.

31.8.7. isAtLeastWindows7 (Class Method)

```
>>--isAtLeastWindows7----------------------------><
```

Returns true if the currently executing program is running on Windows 7, or later. Otherwise it returns false.

31.8.8. isAtLeastXP (Class Method)

```
>>--isAtLeastXP----------------------------------><
```

Returns true if the currently executing program is running on Windows XP, or later. Otherwise it returns false.

31.8.9. isServer2008 (Class Method)

```
>>--isServer2008---------------------------------><
```

Returns true if the currently executing program is running on Windows Server 2008, otherwise false.

31.8.10. isServer2008R2 (Class Method)

```
>>--isServer2008R2-------------------------------><
```

Returns true if the currently executing program is running on Windows Server 2008 Release 2, otherwise false.

31.8.11. isVista (Class Method)

```
>>--isVista--------------------------------------><
```

Returns true if the currently executing program is running on Windows Vista, otherwise false.

31.8.12. isW2K (Class Method)

```
>>--isW2K----------------------------------------><
```

Returns true if the currently executing program is running on Windows 2000, otherwise false.

31.8.13. isW2K3 (Class Method)

```
>>--isW2K3---------------------------------------><
```

Returns true if the currently executing program is running on Windows 2003, otherwise false.

31.8.14. isWindows7 (Class Method)

```
>>--isWindows7-----------------------------------><
```

Returns true if the currently executing program is running on Windows 7, otherwise false.

31.8.15. isWow64 (Class Method)

```
>>--isWow64--------------------------------------><
```

This is an alias for the *is32on64bit*() method. WoW is Microsoft's term for running Windows on Windows. In this case running 32-bit Windows on a 64-bit version of Windows. The method returns true if the currently executing program is running under the 32-bit ooRexx interpreter on a 64-bit version of Windows, otherwise false.

31.8.16. isXP (Class Method)

```
>>--isXP-----------------------------------------><
```

Returns true if the currently executing program is running on Windows XP, otherwise false.

31.8.17. isXP32 (Class Method)

```
>>--isXP-----------------------------------------><
```

Returns true if the currently executing program is running on 32-bit Windows XP, otherwise false.

31.8.18. isXP64 (Class Method)

```
>>--isXP64---------------------------------------><
```

Returns true if the currently executing program is running on 64-bit Windows XP, otherwise false.

31.8.19. settingChanged

```
>>--settingChanged(--+-----------+--+---------+--+--------+--)--><
                     +--timeOut--+  +-,-opts--+  +-,-area-+
```

Sends the WM_SETTINGCHANGE to all top-level windows. Applications should invoke this method when they make changes to system environment parameters.

Arguments:

The arguments are:

timeOut [optional]

Specifies a period in milliseconds to wait when a windows does not respond. The default *timeOut* is 5,000. If the *timeOut* is the special value of 0, the message is sent in a way that does not wait for any window to acknowledge the message and the method returns immediately.

style [optional]

Specifies how the WM_SETTINGCHANGE message is sent. This should be one or more of the following keywords separated by spaces, case is not significant:

ABORTIFHUNG	NORMAL	ERRORONEXIT
BLOCK	NOTIMEOUTIFNOTHUNG	

ABORTIFHUNG

Returns without waiting for the time-out period to elapse if the receiving thread appears to not respond or to be *hung.* This is the default if this argument is omitted.

BLOCK

Prevents the calling thread from processing any other requests until the function returns

NORMAL
: The calling thread is not prevented from processing other requests while waiting for the function to return.

NOTIMEOUTIFNOTHUNG
: Do not enforce the time-out period as long as the receiving thread is processing messages.

ERRORONEXIT
: **Vista or later only.** The broadcast function should return 0 if the receiving window is destroyed or its owning thread dies while the message is being processed.

area [optional]
: This argument indicates the area of change. The default is *Environment* which is correct to indicate the system or user environment variables have changed. The arugment can also be the name of a registry key or the name of a section in the Win.ini file. When the string is a registry name, it typically indicates only the leaf node in the registry, not the full path.

  It is suggested that the programmer should not change this from the default, unless the programmer understands the usage of this argument in the WM_SETTINGCHANGED window message.

Return value:

Returns true on success, false on error.

Remarks:

This method would typically be used to notify the operating system that one of the environment variables, such as the PATH variable, has been changed in the registry. This allows, for instance, updating the path without rebooting the system.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.systemErrorCode*.

Example:

This example updates the user PATH variable and then notifies the system so that the new path takes effect immediately:

```
  reg = .WindowsRegistry~new
  envKey = reg~open(reg~current_user, 'Environment', 'WRITE QUERY')

  -- Error checking code, code that creates the new path string, the data
  -- type for the registry value, etc..

  reg~setValue(envKey, pathName, newPath, valueType)

  reg~close(envKey)


  .OS~settingChanged(2000);

::requires 'ooDialog.cls'
::requires 'winsystm.cls'
```

31.8.20. shellChangeNotify

```
>>--shellChangeNotify(--+---------+-+--------+-+--------+-+--------+--)--------><
                        +-eventID-+ +-,-opt-+ +-,-val1-+ +-,-val2-+
```

Notifies the system of an event that an application has performed. An application should use this function if it performs an action that may affect the Shell. Currently this method can only be used to notify the system that a file type association has changed.

Arguments:

The arguments are:

eventID [reserved optional]

Identifies the event that has occurred. Typically, only one event is specified at a time. If more than one event is specified, the values contained in the *val!* and *val2* arguments must be the same, respectively, for all specified events.

In the current implementation the programmer can not specify the id, it is set to SHCNE_ASSOCCHANGED. If this argument is specified, the method fails.

opts [reserved optional]

A single flag that indicates the meaning of the values contained in the *val1* and *val2* arguments.

In the current implementation the programmer can not specify the flag, it is set to SHCNF_IDLIST. If this argument is specified, the method fails.

val1 [reserved optional]

First event-dependent value.

In the current implementation the programmer can not specify the *val1* value, it is set correctly by the ooDialog framework. If this argument is specified, the method fails.

val [reserved optional]

Second event-dependent value.

In the current implementation the programmer can not specify the *val1* value, it is set correctly by the ooDialog framework. If this argument is specified, the method fails.

Return value:

In the current implementation, returns false if the programmer specifies any of the reserved arguments, otherwise it returns true.

Remarks:

This function notifies the system that a file type association has changed. This allows an application to add, or modify the file type values in the registry and have the changes take effect immediately without rebooting the system. The underlying Windows API that is used in the implementation provides more functionality than this specific notification. Future versions of ooDialog may add that functionality to this method.

Example:

This example writes to the registry to add a file type association and then invokes *shellChangeNotify* to have the new file association take place immediately:

```
reg = .WindowsRegistry~new

-- Code that adds the file type infomation to the registry
...

.OS~shellChangeNotify
```

```
::requires 'ooDialog.cls'
::requires 'winsystm.cls'
```

31.9. Play Routine

This routine is used to play audio sounds.

```
>>-Play(--+----------+--+---------+--)-----------><
          +-fileName-+  +-,--YES--+
                        +-,--LOOP-+
```

The Play routine can be used to play an audio file using the Windows multimedia capabilities. When the arguments are omitted, the currently playing sound file is stopped.

The named file is searched for in the current directory and in the directories of the SOUNDPATH environment variable.

Arguments:

The arguments are:

fileName

The file name of an audio (.WAV) file. The file is searched for in the current directory and in the directories listed in the SOUNDPATH environment variable. If this argument is omitted, the currently playing sound file is stopped.

Note

The system can only play one audio file at a time using the *play*() routine. If an audio file is currently playing, calling the routine with arguments will disrupt the audio file already playing.

option

When this argument is omitted, the audio file is played synchronously. Otherwise, use one of the following keywords:

YES

The audio file is played asynchronously.

LOOP

The audio file is played asynchronously in a loop. The loop is stopped by calling Play again with no arguments.

Example:

The following example plays a welcoming message:

```
rc = play("Welcome.wav")
```

31.10. Point Class

The **`Point`** class represents a point in a 2-D coordinate system.

31.10.1. Method Table

The following table lists the class and instance methods of the **`Point`** class:

Table 31.5. Point Class Method Reference

Method	Description
Class Methods	**Class Methods**
new	Instantiates a new **`Point`** object.
Attributes	**Attributes**
x	Reflects the X coordinate of the point.
y	Reflects the Y coordinate of the point.
Instance Methods	**Instance Methods**
+	Adds two **`Point`**objects
-	Subtracts two **`Point`**objects
copy	Returns a new **`Point`** object that represents the same coordinates as this point.
decr	Decrements the x and y attributes by the amount specified
incr	Increments the x and y attributes by the amount specified
inRect	Tests if this point is within a specified rectangular area.
string	Provides an informative string representation of the **`Point`** object.

31.10.2. new (Class Method)

```
>>--new(--+---+--,--+---+--)---------------------><
          +-x-+     +-y-+
```

Instantiates a new **`Point`** object.

Arguments:

The arguments are:

x [optional]

The x coordinate of the point. The default is 0.

y

The y coordinate of the point. The default is the value of the x coordinate.

Return value:

This method returns a new **`Point`** object.

Remarks:

Both the x and y coordinates of a point are whole numbers in the range of -2147483648 to 2147483647, inclusive. Note that the *x* argument is checked first, if it is omitted, its value is set to 0. The *y*, if omitted, is then set to the value of *x*.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example simply creates a few **`Point`** objects:

```
p = .Point~new(45, 90)
say 'Point p is at ('p~x','p~y')'
say
q = .Point~new(55)
say 'A new point q is at ('q~x','q~y')'
say

q~x = 150
q~y = 300
say 'Changed point q to be at ('q~x','q~y')'
say

::requires 'ooDialog.cls'

/* Output would be:

Point p is at (45,90)

A new point q is at (55,55)

Changed point q to be at (150,300)

*/
```

31.10.3. x (Attribute)

```
>>--x--------------------------------------------><

>>--x=number-------------------------------------><
```

Reflects the x coordinate of the **`Point`**.

x get:
Returns the value of *x* coordinate of the point

x set:
Sets the value of the *x* coordinate of the point. The value must be a whole number in the range of -2147483648 to 2147483647, inclusive.

Details
Raises syntax errors when incorrect arguments are detected.

Example:
See the .Point~new *example*.

31.10.4. y (Attribute)

```
>>--y--------------------------------------------><

>>--y=number-------------------------------------><
```

Reflects the y coordinate of the **`Point`**.

y get:
Returns the value of *y* coordinate of the point

y set:

Sets the value of the *y* coordinate of the point. The value must be a whole number in the range of -2147483648 to 2147483647, inclusive.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

See the .Point~new *example*.

31.10.5. + (addition)

```
>>--"+"--point2---------------------------------><
```

The + method adds the X and Y attributes of this point to another **`Point`**.

Arguments:

The single argument is:

point2 [required]

The **`Point`** object being added.

Return value:

The return value is a **`Point`** object whose X and Y attributes are the sum of the X and Y attributes of this point and the other point.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example shows a generic method used to move a number of controls the same relative distance in a dialog:

```
::method moveControls private
  use strict arg controlArray, x = 0, y = 0

  offSet = .Point~new(x, y)
  do obj over controlArray
    pos = obj~getRealPos
    newPos = pos + offSet
    obj~moveTo(newPos, "NOREDRAW")
  end
```

31.10.6. - (subtraction)

```
>>--"-"--point2---------------------------------><
```

The method subtracts the X and Y attributes of this point from another **`Point`**.

Arguments:

The single argument is:

point2 [required]

The **`Point`** object being subtracted.

Return value:

The return value is a **`Point`** object whose X and Y attributes are the result of subtracting the other point's X and Y attributes from the X and Y attributes of this point.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

A simple example:

```
  p1 = .Point~new(250, 200)
  p2 = .Point~new(25, 20)
  p3 = p1 - p2
  say "Point 3, x:" p3~x "y:" p3~y

  /* Output would be:

     Point 3, x: 225 y: 180

  */
::requires "ooDialog.cls"
```

31.10.7. copy

```
>>--copy-----------------------------------------><
```

Returns a new **`Point`** object that represents the same coordinates as this point.

Arguments:

There are no arguments for this method.

Return value:

Returns a new point whose *x* and *y* attributes are the same as this point's attriubtes.

Example:

This example makes a copy of the current mouse postion so that it can be mapped to the client coordinates of the dialog without altering the position in screen coordinates:

```
  ::method initDialog
    expose mouse

    mouse = .Mouse~new(self)
    ...

::method mouseOverDialog private
  expose mouse

  pos = mouse~getCursorPos
  clientPos = pos~copy

  self~screen2client(clientPos)

  say 'Screen mouse postion:        ' pos
  say 'Dialog client mouse postion:' clientPos
  say 'Mouse over dialog?           ' clientPos~inRect(self~clientRect)

  return clientPos~inRect(self~clientRect)
```

31.10.8. decr

```
>>--decr(--+--------+--+---------+--)------------><
           +--decrX-+  +-,-decrY-+
```

Decrements this point's x and y attributes by the specified amount.

Arguments:

The arguments are:

decrX [optional]

The amount to decrement, (to subtract from,) this point's x attribute. The amount must be a whole number in the range of -2147483648 to 2147483647, inclusive.

If *decrY* is specified and this arg is omitted then the x attribute is not changed. If both *decrX* and *decrY* are ommitted, then the x and y attributes are decremented by 1.

decrY [optional]

The amount to decrement, (to subtract from,) this point's y attribute. The amount must be a whole number in the range of -2147483648 to 2147483647, inclusive.

If *decrX* is specified and this arg is omitted then the y attribute is not changed. If both *decrX* and *decrY* are ommitted, then the x and y attributes are decremented by 1.

Return value:

There is no return.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example changes the pointer to the hourglass shape then moves the cursor one pixel up and one pixel to the left. Moving the cursor forces the cursor to be redrawn

```
oldCursorPosition = self~getCursorPos
oldCursorShape = self~cursor_wait
self~setCursorPos(oldCursorPosition~decr)
```

31.10.9. incr

```
>>--incr(--+--------+--+---------+--)------------><
           +--incrX-+  +-,-incrY-+
```

Increments this point's x and y attributes by the specified amount.

Arguments:

The arguments are:

incrX [optional]

The amount to increment, (to add to,) this point's x attribute. The amount must be a whole number in the range of -2147483648 to 2147483647, inclusive.

If *incrY* is specified and this arg is omitted then the x attribute is not changed. If both *incrX* and *incrY* are ommitted, then the x and y attributes are incremented by 1.

incrY [optional]

The amount to increment, (to add to,) this point's y attribute. The amount must be a whole number in the range of -2147483648 to 2147483647, inclusive.

If *incrX* is specified and this arg is omitted then the y attribute is not changed. If both *incrX* and *incrY* are ommitted, then the x and y attributes are incremented by 1.

Return value:

There is no return.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example changes the pointer to the hourglass shape then moves the cursor one pixel down and one pixel to the right. Moving the cursor it to be redrawn

```
oldCursorPosition = self~getCursorPos
oldCursorShape = self~cursor_wait
self~setCursorPos(oldCursorPosition~incr)
```

31.10.10. inRect

```
>>--inRect(--rect--)----------------------------><
```

Tests if this point is within a specified rectangular area.

Arguments:

The single argument is:

rect [required]

A *Rect* object that specifies the area to be tested.

Return value:

True if the point lies within the rectangle, otherwise false. See the remarks for the definition of *within* .

Remarks:

Windows defines that a point is within a rectangle if it lies on the left or top side or is within all four sides. A point on the right or bottom side is considered outside the rectangle.

The rectangle must be normalized, that is, rect~right must be greater than rect~left and rect~bottom must be greater than rect~top. If the rectangle is not normalized, a point is never considered inside of the rectangle.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example tests if the current cursor position lies with the client area of the dialog:

```
mouse = .Mouse~new(self)
p = mouse~getCursorPos

self~screen2client(p)
```

```
  if p~inRect(self~clientRect) then do
    say 'The mouse cursor is over this dialog'
  end
  else do
    say 'The mouse cursor is NOT over this dialog'
  end
```

31.10.11. string

```
>>--string---------------------------------------><
```

The *string* method over-rides the **.Object** classes *string* method to provide more informative string representation of a **Point** object. The returned string include the value of the x and y attributes.

Arguments:

There are no arguments to this method.

Return value:

Returns a string in the form: *a Point (x, y)* where *x* is the value of the x attribute of the object and *y* is the value of the y attribute.

Example:

A trival example.

```
p = .Point~new(5, 120)
say s

/* Output would be:

a Point (5, 120)

*/
```

31.11. Rect Class

The **Rect** object is most often used to represent a rectangle on a coordinate system, specifically the coordinate system used by Windows where the upper left corner is considered (0,0). However, it is convenient to use the Rect object for other things, for instance the margin around a rectangle.

31.11.1. Method Table

The following table lists the class and instance methods of the **Rect** class:

Table 31.6. Rect Class Method Reference

Method	Description
Class Methods	**Class Methods**
new	Instantiates a new **Rect** object.
Attributes	**Attributes**
bottom	The *bottom* coordinate of the rectangle.
left	The *left* coordinate of the rectangle.

Method	Description
right	The *right* coordinate of the rectangle.
top	The *top* coordinate of the rectangle.
Instance Methods	**Instance Methods**
copy	Returns a new **`Rect`** object with the same coordinates as this rectangle.
string	Provides an informative string representation of the **`Rect`** object.

31.11.2. new (Class Method)

```
>>--new(--+------+--+--------+--+---------+--+----------+--)----><
          +-left-+  +-,-top--+  +-,-right-+  +-,-bottom-+
```

Instantiates a new **`Rect`** object.

Arguments:

The arguments are:

left [optional]

The left coordinate of the rectangle. The default is 0.

top [optional]

The top coordinate of the rectangle. The default is the value of *left*.

right [optional]

The right coordinate of the rectangle. The default is the value of *left*.

bottom [optional]

The bottom coordinate of the rectangle. The default is the value of *left*.

Return value:

The new **`Rect`** object is returned.

Remarks:

All the coordinates of the rectangle are whole numbers in the range of -2147483648 to 2147483647, inclusive. Note that the *left* argument is checked first, if it is omitted, its value is set to 0. All other arguments, if omitted, are set to the value of *left*.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

```
r = .Rect~new(3, 4, 13, 9)
say 'Upper corner ('r~left','r~top') lower corner 'r~right','r~bottom')'
say
margin = .Rect~new(5)
say 'Margins:'
say '  Left: ' margin~left
say '  Top   ' margin~top
say '  Right ' margin~right
say '  Bottom' margin~bottom
say
```

```
r~left = 55
r~top = 75
r~right = 110
r~bottom = 125
say 'New upper corner ('r~left','r~top') new lower corner 'r~right','r~bottom')'
say

/* Output would be:
Upper corner (3,4) lower corner 13,9)

Margins:
  Left:  5
  Top    5
  Right  5
  Bottom 5

New upper corner (55,75) new lower corner 110,125)

/*
```

31.11.3. bottom (Attribute)

```
>>--bottom---------------------------------------><

>>--bottom=number--------------------------------><
```

The *bottom* attribute reflects the y coordinate of the bottom right corner of the rectangle.

bottom get:
Returns the value of the *bottom* coordinate of the rectangle.

bottom set:
Sets the value of the *bottom* coordinate of the rectangle. The value must be a whole number in the range of -2147483648 to 2147483647, inclusive.

Details
Raises syntax errors when incorrect arguments are detected.

Example:
See the .Rect~new *example*.

31.11.4. left (Attribute)

```
>>--left-----------------------------------------><

>>--left=number----------------------------------><
```

The *left* attribute reflects the x coordinate of the upper left corner of the rectangle.

left get:
Returns the value of *left* coordinate of the rectangle.

left set:
Sets the value of the *left* coordinate of the rectangle. The value must be a whole number in the range of -2147483648 to 2147483647, inclusive.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

See the .Rect~new *example*.

31.11.5. right (Attribute)

```
>>--right----------------------------------------><

>>--right=number---------------------------------><
```

The *right* attribute reflects the x coordinate of the bottom right corner of the rectangle.

right get:

Returns the value of *right* coordinate of the rectangle.

right set:

Sets the value of the *right* coordinate of the rectangle. The value must be a whole number in the range of -2147483648 to 2147483647, inclusive.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

See the .Rect~new *example*.

31.11.6. top (Attribute)

```
>>--top------------------------------------------><

>>--top=number-----------------------------------><
```

The *top* attribute reflects the y coordinate of the upper left corner of the rectangle.

top get:

Returns the value of *top* coordinate of the rectangle.

top set:

Sets the value of the *top* coordinate of the rectangle. The value must be a whole number in the range of -2147483648 to 2147483647, inclusive.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

See the .Rect~new *example*.

31.11.7. copy

```
>>--copy-----------------------------------------><
```

Returns a new **`Rect`** object with the same coordinates as this rectangle.

Arguments:

This method has no arguments.

Return value:

Returns a newly instantiated **`Rect`** object with the same coordinates as this object.

Remarks:

The copied rectangle initially has the same coordinates as this rectangle. Changing the coordinates in one rectangle after the copy does not effect the coordinates of the other rectangle.

Example:

This example makes a copy of the client rectangle of an edit control:

```
editOffsetRect = editRect~copy
editOffsetRect~top -= height
editOffSetRect~bottom -= height

say 'text rect:      ' editRect
say 'text offset rect' editOffsetRect

/* Output might be */

text rect:       a Rect (0, 0, 104, 21)
text offset rect a Rect (0, -21, 104, 0)
```

31.11.8. string

```
>>--string---------------------------------------><
```

The *string* method over-rides the **`.Object`** classes *string* method to provide more informative string representation of a **`Rect`** object. The returned string include the value of the coordinates of the rectangle. I.e., the left, top, right, and bottom attributes.

Arguments:

There are no arguments to this method.

Return value:

Returns a string in the form: *a Rect (l, t, r, b)* where *l* is the value of the left attribute, *t* is the value of the top attribute, *r* is the value of the right attribute, and *b* is the value of the bottom attribute of the object.

Example:

A trival example.

```
r = .Rect~new(5, 5, 120, 40)
say r

/* Output would be:

a Rect (5, 5, 120, 40)
```

```
  */
```

31.12. ScreenSize Routine

Obtains the screen size in dialog units and in pixels.

```
>>-ScreenSize()---------------------------------><
```

Argument:

This function takes no arguments.

Return value:

An array containing the screen size. The size in dialog units of the width is at index 1 and the height in dialog units is at index 2. The size in pixels of the width is at index 3 and the height in pixels is at index 4.

Remarks

There are a few problems with this routine. The main problem is with the values returned for the *dialog unit*x. Since dialog units are dependent on the font in use for a specific dialog, and there is no dialog involved here, the routine calculates the dialog units using 8 point System font. It is highly unlikely that this is correct for any specific dialog.

The *screenSize* method of the *DlgUtil* class is a better way to get the screen size. It can provide accurate dialog units and may be more convenient to use since the programmer seldom needs to get the size in both pixels and dialog units.

Example

Get the screen size and display it to the user.

```
size = screenSize()
say
say 'The monitor is' size[1] 'dialog units wide and' size[2] 'dialog units high.'
say 'The monitor is' size[3] 'pixels wide and' size[4] 'pixels high.'
```

31.13. Size Class

A **Size** object encapsulates a width and height dimension in a 2-D coordinate system.

31.13.1. Method Table

The following table lists the class and instance methods of the **Size** class:

Table 31.7. Size Class Method Reference

Method	Description
Class Methods	**Class Methods**
new	Instantiates a new **Size** object.
Attributes	**Attributes**
height	Reflects the height of the **Size** object.
width	Reflects the width of the **Size** object.

Method	Description
Instance Methods	
comparison *Comparison*	Operators that compare one **Size** object to another.
equateTo	Sets the width and height of the **Size** object to that of another **Size** object.
string	Provides an informative string representation of the **Size** object.

31.13.2. new (Class Method)

```
>>-.Size~new(--+-------+--,--+---------+--)------><
               +-width-+     +-height--+
```

Instantiates a new **Size** object with the dimensions specified.

Arguments:

The arguments are:

width

The width for the new object. The default is 0.

height

The height for the new object. The default is the *width* value.

Return value:

This method returns the new **Size** object.

Remarks:

Both the *width* and the *height* of a **Size** object are whole numbers in the range of -2147483648 to 2147483647, inclusive. Note that the *width* argument is checked first, if it is omitted, its value is set to 0. The *height*, if omitted, is then set to the value of *width*.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

```
size = .Size~new(4, 8)
say 'Width:' size~width 'Height:' size~height
say
size2 = .Size~new(16)
say 'A new size object:'
say '  Width: ' size2~width
say '  Height:' size2~height
say

size2~width = 100
size2~height = 300
say 'New width:' size2~width 'new height:' size2~height
say

::requires 'ooDialog.cls'

/* Output would be:

Width: 4 Height: 8
```

```
A new size object:
  Width:  16
  Height: 16

New width: 100 new height: 300

*/
```

31.13.3. height (Attribute)

```
>>--height----------------------------------------><

>>--height=number---------------------------------><
```

Reflects the height of the **`Size`** object.

height get:
Returns the value of *height* of the **`Size`** object.

height set:
Sets the value of the *height* of the **`Size`** object. The value must be a whole number in the range of -2147483648 to 2147483647, inclusive.

Remarks:
Additional comments.

Details
Raises syntax errors when incorrect arguments are detected.

Example:
See the .Size~new *example*.

31.13.4. width (Attribute)

```
>>--width-----------------------------------------><

>>--width=number----------------------------------><
```

Reflects the width of the **`Size`** object.

width get:
Returns the value of *width* of the **`Size`** object.

width set:
Sets the value of the *width* of the **`Size`** object. The value must be a whole number in the range of -2147483648 to 2147483647, inclusive.

Details
Raises syntax errors when incorrect arguments are detected.

Example:
See the .Size~new *example*.

31.13.5. Comparison Operators

```
>>--sizeObj1 =   sizeObj2------------------------><
>>--sizeObj1 ==  sizeObj2------------------------><
>>--sizeObj1 \=  sizeObj2------------------------><
>>--sizeObj1 \== sizeObj2------------------------><
>>--sizeObj1 <   sizeObj2------------------------><
>>--sizeObj1 <<  sizeObj2------------------------><
>>--sizeObj1 <=  sizeObj2------------------------><
>>--sizeObj1 <<= sizeObj2------------------------><
>>--sizeObj1 >   sizeObj2------------------------><
>>--sizeObj1 >>  sizeObj2------------------------><
>>--sizeObj1 >=  sizeObj2------------------------><
>>--sizeObj1 >>= sizeObj2------------------------><
```

The comparsion operators allow one **Size** object to be compared to another **Size** object.

Not every possible comparsion operation is implemented, only the operators listed are supported. All the comparisons follow these two principles. When a comparison operator is doubled, it is a strict comparison. The *width* attribute of each **Size** object is compared and the *height* attribute of each **Size** object is compared. Both comparisons must be true.

When there is not a doubled operator, then the *area* of the size objects is compared. For example:

```
  s1 = .Size~new(4, 4)
  s2 = .Size~new(2, 8)
  s3 = .Size~new(8, 2)
  s4 = .Size~new(2, 8)

  s1 = s2    -- True
  s1 == s2   -- False
  s2 == s4   -- True
  s2 == s3   -- False

  s5 = .Size~new(3, 4)
  s6 = .Size~new(4, 4)

  s5 < s6    -- True
  s5 << s6   -- False
  s5 <= s6   -- True
  s5 <<= s6  -- True
```

The following table lists the comparsion operators. Given two **Size** objects, *s1* and *s2*:

Table 31.8. Size Comparison Operator Reference

Operator	True When
=	(s1~width * s1~height) is equal to (s2~width * s2~height)
==	s1~width is equal to s2~width and s1~height is equal to s2~height.
\=	(s1~width * s1~height) is not equal to (s2~width * s2~height)
\==	s1~width is not equal to s2~width and s1~height is not equal to s2~height.
<	(s1~width * s1~height) is less than (s2~width * s2~height)
<<	s1~width is less than s2~width and s1~height is less than s2~height.
<=	(s1~width * s1~height) is less than or equal to (s2~width * s2~height)
<<=	s1~width is less than or equal to s2~width and s1~height is less than or equal to s2~height.

Operator	True When
>	(s1~width * s1~height) is greater than (s2~width * s2~height)
>>	s1~width is greater than s2~width and s1~height is greater than s2~height.
>=	(s1~width * s1~height) is greater than or equal to (s2~width * s2~height)
>>=	s1~width is greater than or equal to s2~width and s1~height is greater than or equal to s2~height.

31.13.6. equateTo

```
>>--equateTo(--otherSize--)----------------------><
```

The *equateTo* method sets the width and height attributes of this **`Size`** object to the width and height attributes of *otherSize*.

Arguments:

The single argument is:

otherSize

The **`Size`** object that is used to set the *width* and *height* attributes of this object.

Return value:

This method always returns **`.true`**.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This is a simple method as the example shows:

```
  s1 = .Size~new(14, 57)
  s2 = .Size~new(2, 3)

  say 's1:' s1
  say 's2:' s2
  say

  s1~equateTo(s2)

  say 's1:' s1
  say 's2:' s2
  say

  /* Output will be:

s1: a Size (14, 57)
s2: a Size (2, 3)

s1: a Size (2, 3)
s2: a Size (2, 3)

 */
```

31.13.7. string

```
>>--string---------------------------------------><
```

The *string* method over-rides the **.Object** classes *string* method to provide more informative string representation of a **Size** object. The returned string include the value of the width and height attributes.

Arguments:

There are no arguments to this method.

Return value:

Returns a string in the form: *a Size (w, h)* where *w* is the value of the width attribute of the object and *h* is the value of the height attribute.

Example:

A trival example.

```
s = .Size~new(4, 17)
say s

/* Output would be:

a Size (4, 17)

*/
```

31.14. SM Class

The **SM** class (System Metrics) reflects the system metrics or configuration settings of the computer the Rexx program is running on. The class consists entirely of class attributes. For each attribute, the value of the attribute is the value of a system metric or configuration setting.

System metric values and configuration settings can not be changed. The Rexx programmer can not assign a value to any attribute of the **SM** class. A syntax condition is raised if the programmer attempts to change the value of an attribute.

In the **SM** class, all attributes that reflect dimensions are in pixels. All attributes whose attribute name starts with *cx* reflect widths. Likewise, attribute names that start with *cy* reflect heights. System metrics can vary from display to display.

31.14.1. Method Table

The following table lists the class attributes of the **SM** class:

Table 31.9. SM Class Reference

Method	Description
Attributes	**Attributes**
cMouseButtons	The number of buttons on a mouse.
cxCursor	The width of a cursor in pixels.
cxDrag	The width of the drag rectangle in pixels.
cxFixedFrame	The thickness of the border around a non-resizable window that has a caption, in pixels. The width of the vertical border.
cxIcon	The default width of an icon, in pixels.

Method	Description
cxScreen	The width of the primary monitor in pixels.
cxSize	Reflects the width of a button in a window caption or title bar, in pixels.
cxSmIcon	Reflects the recommended width of a small icon, in pixels. Small icons typically appear in window captions and in small icon view.
cxVscroll	The width of a vertical scroll bar in pixels.
cyCaption	The height of a caption area in pixels.
cyCursor	The height of a cursor in pixels.
cyDrag	The height of drag rectangle in pixels.
cyFixedFrame	The thickness of the border around a non-resizable window that has a caption, in pixels. The height of the horizontal border.
cyHscroll	The height of a horizontal scroll bar in pixels.
cyIcon	The default height of an icon, in pixels.
cyMenu	The height of a single-line menu bar, in pixels.
cyScreen	The height of the primary monitor in pixels.
cySize	The height of a button in a window caption or title bar, in pixels.
cySmIcon	The recommended height of a small icon, in pixels.
menuDropAlignment	Reflects whether drop-down menus are right aligned or not.

31.14.2. cMouseButtons (Attribute)

```
>>--cMouseButtons--------------------------------><
```

Reflects the number of buttons on a mouse.

cMouseButtons get:
The number of buttons on a mouse, or zero if no mouse is installed.

cMouseButtons set:
The *cMouseButtons* attribute can not be changed. You can not assign a new value to this attribute.

31.14.3. cxCursor (Attribute)

```
>>--cxCursor-------------------------------------><
```

Reflects the width of a cursor, in pixels.

cxCursor get:
The width of a cursor, in pixels. The system cannot create cursors of other sizes.

cxCursor set:
The *cxCursor* attribute can not be changed. You can not assign a new value to this attribute.

31.14.4. cxDrag (Attribute)

```
>>--cxDrag---------------------------------------><
```

Reflects the width of the drag rectangle in pixels. The drag rectangle is a rectangle that is centered on a drag point to allow for limited movement of the mouse pointer before a drag operation begins. It allows the user to click and release the mouse button easily without unintentionally starting a drag operation.

cxDrag get:
The width of the drag rectangle, in pixels.

cxDrag set:
The *cxDrag* attribute can not be changed. You can not assign a new value to this attribute. However, the size of the drag rectangle can be changed using the *SPI* class through its *dragWidth* and *dragHeight* attributes.

31.14.5. cxFixedFrame (Attribute)

```
>>--cxFixedFrame---------------------------------><
```

Reflects the thickness of a frame around the perimeter of a window that has a caption but is not sizable, in pixels. *cxFixedFrame* is the width of the vertical border and *cyFixedFrame* is the height of the horizontal border.

cxFixedFrame get:
The width of the border (frame) of a non-sizable window that has a caption. This is a normal dialog's border.

cxFixedFrame set:
The *cxFixedFrame* attribute can not be changed. You can not assign a new value to this attribute.

31.14.6. cxIcon (Attribute)

```
>>--cxIcon---------------------------------------><
```

Reflects the default width of an icon, in pixels.

cxIcon get:
The default width of an icon, in pixels.

cxIcon set:
The *cxIcon* attribute can not be changed. You can not assign a new value to this attribute.

31.14.7. cxScreen (Attribute)

```
>>--cxScreen-------------------------------------><
```

Reflects the width of the screen of the primary display monitor, in pixels.

cxScreen get:
The width of the screen in pixels.

cxScreen set:
The *cxScreen* attribute can not be changed. You can not assign a new value to this attribute.

31.14.8. cxSize (Attribute)

```
>>--cxSize---------------------------------------><
```

Reflects the width of a button in a window caption or title bar, in pixels.

cxSize get:
The width of a button in the title bar, such as the close button, in pixels.

cxSize set:
The *cxSize* attribute can not be changed. You can not assign a new value to this attribute.

31.14.9. cxSmIcon (Attribute)

```
>>--cxSmIcon-------------------------------------><
```

Reflects the recommended width of a small icon, in pixels. Small icons typically appear in window captions and in small icon view.

cxSmIcon get:
The recommended width of a small icon, in pixels.

cxSmIcon set:
The *cxSmIcon* attribute can not be changed. You can not assign a new value to this attribute.

31.14.10. cxVscroll (Attribute)

```
>>--cxVscroll------------------------------------><
```

Reflects the width of a vertical scroll bar.

cxVscroll get:
The width of a vertical scroll bar, in pixels.

cxVscroll set:
The *cxVscroll* attribute can not be changed. You can not assign a new value to this attribute.

31.14.11. cyCaption (Attribute)

```
>>--cyCaption------------------------------------><
```

The height of a caption area.

cyCaption get:
  The height of a caption area, in pixels.

cyCaption set:
  The *cyCaption* attribute can not be changed. You can not assign a new value to this attribute.

31.14.12. cyCursor (Attribute)

```
>>--cyCursor-------------------------------------><
```

Reflects the height of a cursor, in pixels.

cyCursor get:
  The height of a cursor, in pixels. The system cannot create cursors of other sizes.

cyCursor set:
  The *cyCursor* attribute can not be changed. You can not assign a new value to this attribute.

31.14.13. cyDrag (Attribute)

```
>>--cyDrag---------------------------------------><
```

Reflects the height of the drag rectangle in pixels. The drag rectangle is a rectangle that is centered on a drag point to allow for limited movement of the mouse pointer before a drag operation begins. It allows the user to click and release the mouse button easily without unintentionally starting a drag operation.

cyDrag get:
  The height of the drag rectangle, in pixels.

cxDrag set:
  The *cyDrag* attribute can not be changed. You can not assign a new value to this attribute. However, the size of the drag rectangle can be changed using the *SPI* class through its *dragWidth* and *dragHeight* attributes.

31.14.14. cyFixedFrame (Attribute)

```
>>--cyFixedFrame---------------------------------><
```

Reflects the thickness of a frame around the perimeter of a window that has a caption but is not sizable, in pixels. *cyFixedFrame* is the height of the horizontal border and *cxFixedFrame* is the width of the vertical border.

cyFixedFrame get:
  The height of the border (frame) of a non-sizable window that has a caption. This is a normal dialog's border.

cxCursor set:

The *cxFixedFrame* attribute can not be changed. You can not assign a new value to this attribute.

31.14.15. cyHscroll (Attribute)

```
>>--cyHscroll------------------------------------><
```

Reflects the height of a horizontal scroll bar.

cyHscroll get:

The height of a horizontal scroll bar, in pixels.

cyHscroll set:

The *cyHscroll* attribute can not be changed. You can not assign a new value to this attribute.

31.14.16. cyIcon (Attribute)

```
>>--cyIcon---------------------------------------><
```

Reflects the default height of an icon, in pixels.

cyIcon get:

The default height of an icon, in pixels.

cyIcon set:

The *cyIcon* attribute can not be changed. You can not assign a new value to this attribute.

31.14.17. cyMenu (Attribute)

```
>>--cyMenu---------------------------------------><
```

Reflects the height of a single-line menu bar.

cyMenu get:

The height of a single-line menu bar, in pixels.

cyMenu set:

The *cyMenu* attribute can not be changed. You can not assign a new value to this attribute.

31.14.18. cyScreen (Attribute)

```
>>--cyScreen-------------------------------------><
```

Reflects the hight of the screen of the primary display monitor, in pixels.

cyScreen get:

The height of the screen in pixels.

cxScreen set:

The *cyScreen* attribute can not be changed. You can not assign a new value to this attribute.

31.14.19. cySize (Attribute)

```
>>--cySize---------------------------------------><
```

Reflects the height of a button in a window caption or title bar, in pixels.

cySize get:

The height of a button in the title bar, such as the close button, in pixels.

cySize set:

The *cySize* attribute can not be changed. You can not assign a new value to this attribute.

31.14.20. cySmIcon (Attribute)

```
>>--cySmIcon-------------------------------------><
```

Reflects the recommended height of a small icon, in pixels. Small icons typically appear in window captions and in small icon view.

cySmIcon get:

The recommended height of a small icon, in pixels.

cySmIcon set:

The *cySmIcon* attribute can not be changed. You can not assign a new value to this attribute.

31.14.21. menuDropAlignment (Attribute)

```
>>--menuDropAlignment-------------------------------------------><
```

Reflects whether drop-down menus are right aligned or not.

menuDropAlignment get:

The value of the attribute is nonzero if drop-down menus are right-aligned with the corresponding menu-bar item, and 0 if the menus are left-aligned.

menuDropAlignment set:

The *menuDropAlignment* attribute can not be changed. You can not assign a new value to this attribute.

31.15. SPI Class

The **`SPI`** class (System Parameters Information) reflects the system-wide parameters of the computer the Rexx program is running on. The class mostly consists of class attributes. For each attribute, the value of the attribute is the value of a system parameter. Changing the value of the attribute changes the value of the system parameter.

When a system-wide parameter is set, the operating system will optionally update the user profile and additionally broadcast the WM_SETTINGCHANGE message to all top-level windows to notify them of the change. The programmer can specify to do neither, update the profile only, or update the profile and broadcast the message. Specifying the action to take is done through the *updateFlag* attribute.

31.15.1. Method Table

The following table lists the class attributes of the **`SPI`** class:

Table 31.10. SPI Class Reference

Method	Description
Constants	**Constants**
WHEEL_PAGESCROLL	Indicates a turn of the mouse wheel should scroll 1 page rather than some number of lines.
Attributes	**Attributes**
dragHeight	Reflects the height of the drag rectangle.
dragWidth	Reflects the width of the drag rectangle.
menuAnimation	Reflects whether menu animation is enabled or disabled.
menuFade	Reflects whether menu animation uses fade or slide animation.
mouseHoverHeight	Reflects the height in pixels of the rectangle the mouse needs to stay in to generate a hover notification.
mouseHoverTime	Reflects the time in milliseconds the mouse has to remain in the hover rectangle to generate a hover notification.
mouseHoverWidth	Reflects the width in pixels of the rectangle the mouse needs to stay in to generate a hover notification.
updateFlag	Defines additional actions to take when a system-wide parameter is set.
wheelScrollLines	Reflects the number of lines to scroll when the vertical mouse wheel is moved.
workArea	Reflects the size of the work area of the screen.

31.15.2. WHEEL_PAGESCROLL (Constant)

```
>>--val=.SPI~WHEEL_PAGESCROLL--------------------><
```

WHEEL_PAGESCROLL is a value defined by the Windows *SDK* that indicates a turn of one notch in the mouse wheel should scroll one page rather than some number of lines. See the *wheelScrollLines* attribute more more information on the mouse wheel.

31.15.3. dragHeight (Attribute)

```
>>--dragHeight----------------------------------><

>>--dragHeight=height---------------------------><
```

Reflects the height of the drag rectangle. The drag rectangle is the rectangle used to detect the start of a drag operation.

dragHeight get:

Retrieves the height of the system wide drag rectangle in pixels.

dragHeight set:

Use a non-negative whole number to set the height, in pixels, of the drag rectangle.

Remarks:

The drag rectangle is a rectangle that is centered on a drag point to allow for limited movement of the mouse pointer before a drag operation begins. It allows the user to click and release the mouse button easily without unintentionally starting a drag operation.

Details

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect usage is detected.

31.15.4. dragWidth (Attribute)

```
>>--dragWidth------------------------------------><

>>--dragWidth=width------------------------------><
```

Reflects the width of the drag rectangle. The drag rectangle is the rectangle used to detect the start of a drag operation.

dragWidth get:

Retrieves the width of the system wide drag rectangle in pixels.

dragWidth set:

Use a non-negative whole number to set the width, in pixels, of the drag rectangle.

Remarks:

The drag rectangle is a rectangle that is centered on a drag point to allow for limited movement of the mouse pointer before a drag operation begins. It allows the user to click and release the mouse button easily without unintentionally starting a drag operation.

Details

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect usage is detected.

31.15.5. menuAnimation (Attribute)

```
>>--menuAnimation--------------------------------><

>>--menuAnimation = onOff------------------------><
```

Reflects whether menu animation is on or off.

menuAnimation get:

The value of the attribute is `.true` if menu animation is on and `.false` when menu animation is off.

menuAnimation set:
Enables or disables menu animation. Set the attribute to **`.true`** to enable menu animation and to **`.false`** to disable menu animation.

Remarks:
Menu animation must be on for *any* menu animations to occur. The *menuFade* attribute reflects whether menus use slide or fade animation.

Details
Sets the *.systemErrorCode*.

Raises syntax errors when incorrect usage is detected.

31.15.6. menuFade (Attribute)

```
>>--menuFade-------------------------------------><

>>--menuFade = onOff-----------------------------><
```

Reflects whether menu animation uses fade or slide animation.

menuFade get:
The value of the attribute is **`.true`** if menu fade animation is used and **`.false`** when menu fade animation is not used.

menuFade set:
Enables or disables menu fade animation. Set the attribute to **`.true`** to enable menu fade animation and to **`.false`** to disable menu fade animation.

Remarks:
When menu fade is off, menus use slide animation. For any animation to be used, *menuAnimation* must be on. Menu fade effect is only possible if the system has a color depth of more than 256 colors.

Details
Sets the *.systemErrorCode*.

Raises syntax errors when incorrect usage is detected.

31.15.7. mouseHoverHeight (Attribute)

```
>>--mouseHoverHeight-----------------------------><

>>--mouseHoverHeight=height----------------------><
```

Reflects the height of the hover rectangle. The hover rectangle is the rectangle within which the mouse pointer has to stay to generate a MOUSEHOVER notification.

mouseHoverHeight get:
Retrieves the height of the system wide hover rectangle in pixels.

mouseHoverHeight set:
Sets the width, in pixels, of the hover rectangle. The *width* must be a non-negative whole number.

Remarks:

The hover rectangle is the rectangle within which the mouse pointer has to stay for the operating system to generate a *connectEvent* notification. Normally the mouse hover event is not generated. The *trackEvent* method must be invoked to request hover tracking before the operating system will generate the hover notification.

Details

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect usage is detected.

31.15.8. mouseHoverTime (Attribute)

```
>>--mouseHoverTime-------------------------------><

>>--mouseHoverTime=milliseconds------------------><
```

Reflects the time, in milliseconds the mouse must remain in the hover rectangle to generate a MOUSEHOVER notification.

mouseHoverTime get:

Retrieves the system wide default hover time out value in milliseconds

mouseHoverTime set:

Sets the time, in milliseconds, for the hover time out. The *milliseconds* must be a non-negative whole number within the range of 10 to 2147483647.

Remarks:

The hover rectangle is the rectangle within which the mouse pointer has to stay for the operating system to generate a *connectEvent* notification. The time out value is the number of milliseconds the mouse has to remain in the rectangle. Normally the mouse hover event is not generated. The *trackEvent* method must be invoked to request hover tracking before the operating system will generate the hover notification.

Details

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect usage is detected.

31.15.9. mouseHoverWidth (Attribute)

```
>>--mouseHoverWidth------------------------------><

>>--mouseHoverWidth=width------------------------><
```

Reflects the width of the hover rectangle. The hover rectangle is the rectangle within which the mouse pointer has to stay to generate a MOUSEHOVER notification.

mouseHoverWidth get:

Retrieves the width of the system wide hover rectangle in pixels.

mouseHoverWidth set:

Sets the width, in pixels, of the hover rectangle. The *width* must be a non-negative whole number.

Remarks:

The hover rectangle is the rectangle within which the mouse pointer has to stay for the operating system to generate a *connectEvent* notification. Normally the mouse hover event is not generated. The *trackEvent* method must be invoked to request hover tracking before the operating system will generate the hover notification.

Details

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect usage is detected.

31.15.10. updateFlag (Attribute)

```
>>--updateFlag-----------------------------------><

>>--updateFlag=newFlag---------------------------><
```

The *updateFlag* attribute allows the programmer to direct the operating system to update the user profile and to broadcast the WM_SETTINGCHANGE message when a system parameter value is changed through the **SPI** class.

updateFlag get:

Retrives the current value of the *updateFlag* attribute. The value is returned as a key word and will be exactly one of these words: UpdateProfile SendChange None. The meaning of the key words is explained below.

updateFlag set:

Assign a key word to the *updateFlag* attribute to specify additional actions for the operating system to take when changing a system parameter through the **SPI** class.

The key word must be exactly one of the following key words, case is not significant and only the first letter of the word needs to be specified:

None	UpdateProfile	SendChange

None

The operating system takes no additional action when the value of one of the system parameter attributes is changed through the **SPI** class.

UpdateProfile

When a system parameter attribute's value is changed through the **SPI** class, the operating system will also update the user profile.

SendChange

When a system parameter attribute's value is changed through the **SPI** class, the operating system updates the user profile and broadcasts the WM_SETTINGCHANGE message to all top-level windows to notify them of the change.

Remarks:

The value of the *updateFlag* attribute only comes into play when a new value is assigned to a system parameter attribute. *updateFlag* is ignored when the value of a system parameter is retrieved. By default, *updateFlag* is set to None.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example changes the WheelScrollLines system parameter to 5. It has the operating system update the user profile and broadcast the setting changed message to all top-level windows so the applications currently running can update their settings if needed. It then changes the *updateFlag* back to the default.

```
  .SPI~updateFlag = 'SendChange'
  .SPI~wheelScrollLines = 5
  .SPI~updateFlag = 'N'
```

31.15.11. wheelScrollLines (Attribute)

```
>>--wheelScrollLines-----------------------------><

>>--wheelScrollLines=count-----------------------><
```

Reflects the number of lines to scroll when the vertical mouse wheel is moved. The number is a non-negative whole number. The system default is 3.

wheelScrollLines get:

Retrieves the number of lines to scroll when the vertical mouse wheel is moved.

wheelScrollLines set:

Sets the number of lines to scroll when the vertical mouse wheel is moved.

Remarks:

The wheel scroll lines is the suggested number of lines to scroll when the mouse wheel is rolled without using modifier keys. If the number is 0, then no scrolling should occur. If the number of lines to scroll is larger than the number of viewable lines, and especially if it is the **`SPI`** constant WHEEL_PAGESCROLL (**`.SPI~WHEEL_PAGESCROLL`**), the scroll operation should be interpreted as clicking once in the page down or page up regions of the scroll bar.

Details

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect arguments are detected.

Example:

This example retrieves the current setting for the wheel scroll lines and then scrolls the text in an edit control the appropriate amount. Note that this is a simplistic example. The code should really check if *scrollLines* is 0, or equal to the WHEEL_PAGESCROLL constant, or greater than the number of visible lines in the edit control.

```
::method scrollUp private
  use strict arg editCtrl

  scrollLines = .SPI~wheelScrollLines
  editCtrl~scrollCommand("UP", scrollLines)
```

31.15.12. workArea (Attribute)

```
>>--workArea-------------------------------------><
```

```
>>--workArea=rArea-------------------------------><
```

Reflects the size of the work area on the screen as a **`.Rect`** object. The work area is the portion of the screen not hidden by the system taskbar or by application desktop toolbars.

workArea get:

The value of the *workArea* attribute is a *Rect* object.

workArea set:

Assign a **`.Rect`** object as the value of this attribute. This changes the current work area to the coordinates specified in the **`.Rect`** object.

Remarks:

When setting a new work area, the **`.Rect`** object specifies the new work area in virtual screen coordinates. If the system has multiple display monitors, the operating system sets the work area of the monitor that contains the specified rectangle.

Details

Sets the *.systemErrorCode*.

Raises syntax errors when incorrect usage is detected.

31.16. .systemErrorCode object

The .systemErrorCode can be used, at times, to obtain additional information when the invocation of a method generates an operating system error.

ooDialog provides an interface to the Windows operating system APIs. To be specific, the Win32 APIs, and mostly those dealing with dialogs and other graphical aspects of the user interface. Some of the Win32 APIs will set a system error code when something goes wrong. (On the other hand, many of the Win32 APIs do not set a system error code.)

When the ooDialog framework first initializes, it sets the value of .systemErrorCode to 0. *Some*, of the methods of the ooDialog classes will set the .systemErrorCode to the value of the system error code when the method detects that a Win32 API has failed and has set the error code. Most methods do **not** change the value of .SystemErrorCode. Only those methods that explicitly document they use .systemErrorCode will ever change its value.

One consequence of the fact that most methods do not change .systemErrorCode is that its value is only meaningful immediately after the invocation of a method that is documented as using .SystemErrorCode. Those methods will set the code to 0 and, if they detect a Win32 API has set the system error code to non-zero, they will then set .systemErrorCode to the value of the system error. Checking .systemErrorCode after the invocation of a method that does not set .systemErrorCode will tell the programmer nothing. Neither that a system error happened, or that it did not happen.

The *getImage* method of the *Image* class is one method that does change .SystemErrorCode. **Note** in the example the use of the RexxUtil function, **`SysGetErrorText`**. This function is useful to get an explanatory message for the error code. The message is provided by Windows, not by ooRexx or ooDialog. This is an example of a possible usage:

```
image = .Image~getImage("resources\bogus.bmp")
if image~isNull then do
  say "Error getting the bogus.bmp image."
  if .systemErrorCode <> 0 then do
```

```
    say 'System error code:' .systemErrorCode
    say '  system message:' SysGetErrorText(.systemErrorCode)
  end
end

/*
  Output on the screen might be:

Error getting the bogus.bmp image.
System error code: 2
  system message: The system cannot find the file specified.
*/
```

31.17. VK Class

The **VK** class allows the programmer to use symbolic names in a program instead of the numeric value of the virtual key codes.

The Windows operating system assigns a value to each key on the keyboard called a scan code. Scan codes are device *dependent*. To notify an application that the user has pressed a key, the operating system translates the scan code into a *virtual key code*. Virtual key codes are device *independent*. This allows the programmer to write a program without worrying about what keyboard the user may have.

Virtual key codes are in the range of 0 through 255. However a number of the possible codes are unassigned, perhaps 30 or 40. The **VK** class provides constants for every assigned key code. The constants serve as symbolic names for the key codes. In addition, the *key2name* method is provided that translates a numeric number into its human-readable string name.

There are two ways to use the **VK** class. The class is a *mixinclass*. It can be inherited by a class to provide that class with all the constants and methods of the **VK** class. Recall that the *::constant* directive defines both a class and an instance method that returns a constant value. Therefore, the constant values can be accessed directly through the class methods of the **VK** class. There is also a *key2name* class method along with the instance method. The following two examples should clarify that:

```
::class 'SimpleDlg' subclass RcDialog inherit VK

::method onKeyDown
  use arg controlID, key

  say "The key pressed was the" self~name2key(key) "virtual key"

  if key == self~DELETE then do
    return self~deleteAllItems
  end
```

The above example inherits the **VK** class and thus the DELETE constant and the *key2name* method become part of the **SimpleDlg** class. The same thing can be accomplished using the class methods of the **VK** class:

```
::class 'SimpleDlg' subclass RcDialog

::method onKeyDown
  use arg controlID, key

  say "The key pressed was the" .VK~name2key(key) "virtual key"
  if key == .VK~DELETE then do
    return self~deleteAllItems
  end
```

Using the class methods of the **VK** class has the advantage of not conflicting with any existing method the **SimpleDlg** may have.

31.17.1. Method Table

The **VK** class primarily consists of constants. The implemented constants and methods are listed in the following table.

Table 31.11. Methods of the VK class

Method	Description
Class Methods	**Class Methods**
key2name	Returns the symbolic name for a numeric key code as a string
Instance Methods	**Instance Methods**
key2name	Returns the symbolic name for a numeric key code as a string
Constants	**Constants**
0	The **0** key
1	The **1** key
2	The **2** key
3	The **3** key
4	The **4** key
5	The **5** key
6	The **6** key
7	The **7** key
8	The **8** key
9	The **9** key
A	The **A** key
B	The **B** key
C	The **C** key
D	The **D** key
E	The **E** key
F	The **F** key
G	The **G** key
H	The **H** key
I	The **I** key
J	The **J** key
K	The **K** key
L	The **L** key
M	The **M** key
N	The **N** key
O	The **O** key

Method	Description
P	The **P** key
Q	The **Q** key
R	The **R** key
S	The **S** key
T	The **T** key
U	The **U** key
V	The **V** key
W	The **W** key
X	The **X** key
Y	The **Y** key
Z	The **Z** key
ACCEPT	The IME accept
ADD	The Add key
APPS	The Applications key (Natural keyboard)
ATTN	The Attn key
BACK	The BACKSPACE key
BROWSER_BACK	The Browser Back key
BROWSER_FAVORITES	The Browser Favorites key
BROWSER_FORWARD	The Browser Forward key
BROWSER_HOME	The Browser Start and Home key
BROWSER_REFRESH	The Browser Refresh key
BROWSER_SEARCH	The Browser Search key
BROWSER_STOP	The Browser Stop key
CANCEL	The Control-break processing
CAPITAL	The CAPS LOCK key
CLEAR	The CLEAR key
CONTROL	The CTRL key
CONVERT	The IME convert
CRSEL	The CrSel key
DECIMAL	The Decimal key
DELETE	The DEL key
DIVIDE	The Divide key
DOWN	The DOWN ARROW key
END	The END key
EREOF	The Erase EOF key
ESCAPE	The ESC key
EXECUTE	The EXECUTE key
EXSEL	The ExSel key

Method	Description
F1	The F1 key
F2	The F2 key
F3	The F3 key
F4	The F4 key
F5	The F5 key
F6	The F6 key
F7	The F7 key
F8	The F8 key
F9	The F9 key
F10	The F10 key
F11	The F11 key
F12	The F12 key
F13	The F13 key
F14	The F14 key
F15	The F15 key
F16	The F16 key
F17	The F17 key
F18	The F18 key
F19	The F19 key
F20	The F20 key
F21	The F21 key
F22	The F22 key
F23	The F23 key
F24	The F24 key
FINAL	The IME final mode
HANGUL	The IME Hangul mode
HANJA	The IME Hanja mode
HELP	The HELP key
HOME	The HOME key
ICO_00	The 00 key on ICO
ICO_CLEAR	No explanation
ICO_HELP	The Help key on ICO
INSERT	The INS key
JUNJA	The IME Junja mode
KANA	The IME Kana mode
KANJI	The IME Kanji mode
LAUNCH_APP1	The Start Application 1 key
LAUNCH_APP2	The Start Application 2 key

Method	Description
LAUNCH_MAIL	The Start Mail key
LAUNCH_MEDIA_SELECT	The Select Media key
LBUTTON	The Left mouse button
LCONTROL	The Left CONTROL key
LEFT	The LEFT ARROW key
LMENU	The Left MENU key
LSHIFT	The Left SHIFT key
LWIN	The Left Windows key (Natural keyboard)
MBUTTON	The Middle mouse button (three-button mouse)
MEDIA_NEXT_TRACK	The Next Track key
MEDIA_PLAY_PAUSE	The Play/Pause Media key
MEDIA_PREV_TRACK	The Previous Track key
MEDIA_STOP	The Stop Media key
MENU	The ALT key
MODECHANGE	The IME mode change request
MULTIPLY	The Multiply key
NEXT	The PAGE DOWN key
NONCONVERT	The IME nonconvert
NUMLOCK	The NUM LOCK key
NUMPAD0	The Numeric keypad 0 key
NUMPAD1	The Numeric keypad 1 key
NUMPAD2	The Numeric keypad 2 key
NUMPAD3	The Numeric keypad 3 key
NUMPAD4	The Numeric keypad 4 key
NUMPAD5	Numeric keypad 5 key
NUMPAD6	The Numeric keypad 6 key
NUMPAD7	The Numeric keypad 7 key
NUMPAD8	The Numeric keypad 8 key
NUMPAD9	The Numeric keypad 9 key
OEM_1	Used for miscellaneous characters; it can vary by keyboard. For the US standard keyboard, the ';:' key
OEM_2	Used for miscellaneous characters; it can vary by keyboard. For the US standard keyboard, the '/?' key
OEM_3	Used for miscellaneous characters; it can vary by keyboard. For the US standard keyboard, the '`~' key
OEM_4	Used for miscellaneous characters; it can vary by keyboard. For the US standard keyboard, the '[{' key
OEM_5	Used for miscellaneous characters; it can vary by keyboard. For the US standard keyboard, the '\\|' key

Method	Description
OEM_6	Used for miscellaneous characters; it can vary by keyboard. For the US standard keyboard, the ']}' key
OEM_7	Used for miscellaneous characters; it can vary by keyboard. For the US standard keyboard, the 'single-quote/double-quote' key
OEM_8	Used for miscellaneous characters; it can vary by keyboard.
OEM_102	Either the angle bracket key or the backslash key on the RT 102-key keyboard
OEM_ATTN	Nokia/Ericsson definition
OEM_AUTO	Nokia/Ericsson definition
OEM_AX	The *AX* key on Japanese AX kbd
OEM_BACKTAB	Nokia/Ericsson definition
OEM_CLEAR	The Clear key
OEM_COMMA	For any country/region, the ',' key
OEM_COPY	Nokia/Ericsson definition
OEM_CUSEL	Nokia/Ericsson definition
OEM_ENLW	Nokia/Ericsson definition
OEM_FINISH	Nokia/Ericsson definition
OEM_JUMP	Nokia/Ericsson definition
OEM_MINUS	For any country/region, the '-' key
OEM_PA1	Nokia/Ericsson definition
OEM_PA2	Nokia/Ericsson definition
OEM_PA3	Nokia/Ericsson definition
OEM_PERIOD	For any country/region, the '.' key
OEM_RESET	Nokia/Ericsson definition
OEM_WSCTRL	Nokia/Ericsson definition
PACKET	Used to pass Unicode characters as if they were keystrokes. The PACKET key is the low word of a 32-bit Virtual Key value used for non-keyboard input methods.
OEM_PLUS	For any country/region, the '+' key
PA1	The PA1 key
PAUSE	The PAUSE key
PLAY	The Play key
PRINT	The PRINT key
PRIOR	The PAGE UP key
PROCESSKEY	The IME PROCESS key
RBUTTON	The Right mouse button
RCONTROL	The Right CONTROL key
RETURN	The ENTER key
RIGHT	The RIGHT ARROW key
RMENU	The Right MENU key

Method	Description
RSHIFT	The Right SHIFT key
RWIN	The Right Windows key (Natural keyboard)
SCROLL	The SCROLL LOCK key
SELECT	The SELECT key
SEPARATOR	The Separator key
SHIFT	The SHIFT key
SLEEP	The Computer Sleep key
SNAPSHOT	The PRINT SCREEN key
SPACE	The SPACEBAR
SUBTRACT	The Subtract key
TAB	The TAB key
UP	The UP ARROW key
VOLUME_DOWN	The Volume Down key
VOLUME_MUTE	The Volume Mute key
VOLUME_UP	The Volume Up key
XBUTTON1	The X1 mouse button
XBUTTON2	The X2 mouse button
ZOOM	The Zoom key

31.17.2. key2name

```
>>--key2name(--numericCode--)--------------------><
```

Converts a numeric virtual key code into the string version of its constant name.

Arguments:

The only argument is:

numericCode [required]

The numeric value of a virtual key code. Virtual key codes are whole numbers in the range 0 through 255.

Return value:

Returns the constant method name for the specified virtual key code as a string.

Remarks:

There is both a class method and an instance method with this method's name. Therefore the method can be use like this:

```
::class 'MyClass' subclass ResDialog inherit VK
  ...
  say "The key pressed was:" self~key2Name(key)
```

or like this:

```
::class 'MyClass' subclass RcDialog
```

```
    ...
    say "The key pressed was:" .VK~key2name(key)
```

Not all possible key codes are assigned. The empty string is returned for those numbers.

This method is a convenience method which allows a program to print the key code name to the screen in a say statement. The method is usually not needed. In particular, code like this:

```
  ::method someMethod
    use arg key
    ...
    if .VK~key2Name(key) == "INSERT" then
      ...
```

can be replaced by this:

```
  ::method someMethod
    use arg key
    ...
    if key == .VK~INSERT then
      ...
```

Details

A syntax error is raised if *numericCode* is not a whole number the range of 0 through 255.

31.17.3. Example VK Constant Use

This example connects the key down event in a tree control with a method in the dialog. If the user presses the delete key while the tree control has the focus, the selected item is deleted. Likewise, if the user presses the insert key he is given the opportunity to insert a new item into the tree.

```
::class 'TreeDlg' subclass ResDialog
...

::method initDialog

self~connectTreeViewEvent(IDC_TREE, "KEYDOWN", onKeyDown)
...

::method onKeyDown
use arg treeId, key
tree = self~newTreeView(treeId)

-- If the delete key is pressed, delete the selected item, or if the insert
-- key is pressed insert a new item by programmatically clicking the New Item
-- button. Otherwise ignore key.
if key == .VK~DELETE then
   tree~delete(tree~selected)
else if key == .VK~INSERT then
   self~newPushButton(IDC_PB_NEW_ITEM)~push
```

31.18. Window Class

A *Window* object allows the invocation of methods common to every window. It does this by inheriting the *WindowBase* class. Other than the *new*() method, the *Window* class provides no other methods of its own.

31.18.1. Method Table

The following table lists the class and instance methods of the **`Window`** class:

Table 31.12. Window Class Method Reference

Window Method	Description
Class Methods	**Class Methods**
new	Instantiates a new window object.
Attributes	**Attributes**
factorX	The horizontal size of one dialog unit in pixels. (Inaccurate.)
factorY	The vertical size of one dialog unit in pixels. (Inaccurate.)
hwnd	The window handle of the window.
initCode	Has no meaning, always 0.
pixelCX	The width of the window in pixels.
pixelCY	The height of the window in pixels.
sizeX	The width of the window in dialog units. (Inaccurate.)
sizeY	The height of the window in dialog units. (Inaccurate.)
Instance Methods	
clear	Clears the client area of the window by painting it with the background brush.
client2screen	Converts a point or points in client-area coordinates of the window to its screen coordinates.
clientRect	Returns a **`Rect`** object containing the dimensions of the window's client area in pixels.
clientToScreen	Converts client-area coordinates of the window to its screen coordinates.
disable	Disables the window.
display	Shows or hides the window.
draw	Redraws the entire client area of the window immediately.
enable	Enables the window.
foregroundWindow	Returns the handle of the window in the foreground.
getClientRect	Returns the dimensions of the window's client area.
getExStyleRaw	Retrieves the numeric value of the window's extended style flags.
getID	Retrieves the identification number of the window.
getPos	Returns the position of the window in dialog units **(not accurate.)**
getRealPos	Returns the position of the window in pixels as a **`Point`** object.
getRealSize	Returns the size of the window in pixels as a **`Size`** object.
getRect	Returns the dimensions of the window.
getSize	Returns the size of the window in dialog units **(not accurate.)**
getStyleRaw	Retrieves the numeric value of the window's style flags.
getText	Gets the text of the window.
getTextSizePx	Calculates the size needed for a string in pixels **(preferred method.)**
getTextSizeScreen	Calculates the size needed for a string in pixels.

Window Method	Description
hide	Makes the window invisible and repaints it.
hideFast	Marks the window as invisible
isEnabled	Tests if the window is enabled.
isVisible	Tests if the window is visible.
move	Moves the window to the position specified in dialog units **(not accurate.)**
moveTo	Moves the window to the position specified in pixels.
redraw	Redraws the entire window and all its child windows immediately.
redrawClient	Redraws the entire client area of the window immediately.
resize	Resizes the window to the size specified in dialog units **(not accurate.)**
resizeTo	Resizes the window to the size specified in pixels.
screen2client	Converts a point or points in screen coordinates to the client-area coordinates of the window.
screenToClient	Converts screen coordinates to the client-area coordinates of the window.
sendMessage	Sends a Windows message to the underlying window and returns its response as a whole number.
sendMessageHandle	Sends a Windows message to the underlying window and returns its response as a handle.
setRect	Moves and / or resizes the window.
setStyleRaw	Sets the value of the window's style flags using the numeric value specified.
setText	Sets the text of the window.
setTitle	Sets the text of the window.
showFast	Marks the window as visible.
title	Gets the text of the window.
title=	Sets the text of the window.
update	Invalidates the entire client area of the window.
windowRect	Returns a **`Rect`** object containing the dimensions of the window in pixels.

31.18.2. new (Class Method)

```
>>--new(--hwnd--)--------------------------------><
```

Given a window *handle*, instantiates a new **`Window`** object.

Arguments:

The single argument is:

hwnd

The window

Return value:

This method returns a new **`Window`** object.

Example:

Given an array of dialog control resource IDs this method enables all the controls without having to know what the type of the individual control is:

```
::method enableControls
  use strict arg IDs
  hDlg = self~hwnd
  do id over IDs
    hwnd = self~getControlHandle(id, hDlg)
    if hwnd = 0 then return .false
    .Window~new(hwnd)~enable
  end
  return .true
```

31.18.3. initCode (Attribute)

```
WindowBase::initCode


>>--initCode------------------------------------><

>>--initCode=code-------------------------------><
```

31.18.4. hwnd (Attribute)

```
WindowBase::hwnd


>>--hwnd----------------------------------------><
```

31.18.5. factorX (Attribute)

```
WindowBase::factorX


>>--factorX-------------------------------------><

>>--factorX=ratio-------------------------------><
```

31.18.6. factorY (Attribute)

```
WindowBase::factorY


>>--factorY-------------------------------------><

>>--factorY=ratio-------------------------------><
```

31.18.7. pixelCX (Attribute)

```
WindowBase::pixelCX
```

```
>>--pixelCX--------------------------------------><
```

31.18.8. pixelCY (Attribute)

```
WindowBase::pixelCY


>>--pixelCY--------------------------------------><
```

31.18.9. sizeX (Attribute)

```
WindowBase::sizeX


>>--sizeX----------------------------------------><

>>--sizeX=dialogUnits----------------------------><
```

31.18.10. sizeY (Attribute)

```
WindowBase::sizeY


>>--sizeY----------------------------------------><

>>--sizeY=dialogUnits----------------------------><
```

31.18.11. clear

```
WindowBase::clear


>>--clear----------------------------------------><
```

31.18.12. client2screen

```
WindowBase::client2screen


>>--client2screen(--pointOrRect--)---------------><
```

31.18.13. clientRect

```
WindowBase::clientRect


>>--clientRect(--+--------+--)-------------------><
```

```
                  +--hwnd--+
```

31.18.14. clientToScreen

```
WindowBase::clientToScreen


>>--clientToScreen(--x--,--y--)------------------><
```

31.18.15. disable

```
WindowBase::disable


>>--disable--------------------------------------><
```

31.18.16. display

```
WindowBase::display
```

31.18.17. draw

```
WindowBase::draw


>>--draw-----------------------------------------><
```

31.18.18. enable

```
WindowBase::enable


>>--enable---------------------------------------><
```

31.18.19. foregroundWindow

```
WindowBase::foregroundWindow


>>--foregroundWindow-----------------------------><
```

31.18.20. getClientRect

```
WindowBase::getClientRect


>>--getClientRect(--+------+--)------------------><
                    +-hwnd-+
```

31.18.21. getExStyleRaw

```
WindowBase::getExStyleRaw


>>--getExStyleRaw--------------------------------><
```

31.18.22. getID

```
WindowBase::getID


>>--getID----------------------------------------><
```

31.18.23. getPos

```
WindowBase::getPos


>>--getPos---------------------------------------><
```

31.18.24. getRealPos

```
WindowBase::getSize


>>--getRealPos-----------------------------------><
```

31.18.25. getRealSize

```
WindowBase::getRealSize


>>--getRealSize----------------------------------><
```

31.18.26. getRect

```
WindowBase::getRect


>>--getRect--------------------------------------><
```

31.18.27. getSize

```
WindowBase::getSize


>>--getSize--------------------------------------><
```

31.18.28. getStyleRaw

```
WindowBase::getStyleRaw


>>--getStyleRaw----------------------------------><
```

31.18.29. getText

```
WindowBase::getText


>>--getText--------------------------------------><
```

31.18.30. getTextSizePx

```
WindowBase::getTextSizePx


>>--getTextSizePx(-text--)-----------------------><
```

31.18.31. getTextSizeScreen

```
WindowBase::getTextSizeScreen


>>--getTextSizeScreen(-text--+---------+--+------------+--+-------------+-)----><
                             +-,-type--+  +-,-fontSrc--+  +-,-fontSize--+
```

31.18.32. hide

```
WindowBase::hide


>>--hide-----------------------------------------><
```

31.18.33. hideFast

```
WindowBase::hideFast


>>--hideFast-------------------------------------><
```

31.18.34. isEnabled

```
WindowBase::isEnabled


>>--isEnabled------------------------------------><
```

31.18.35. isVisible

```
WindowBase::isVisible


>>--isVisible------------------------------------><
```

31.18.36. move

```
WindowBase::move


>>--move(--xPos--,--yPos--+-------------+--)-----><
                          +-,-showOpts--+
```

31.18.37. moveTo

```
WindowBase::moveTo


Form 1:

>>--moveTo(--point--+--------------+--)----------><
                    +--,-showOpts--+
Form 2:

>>--moveTo(--x,--y--+--------------+--)----------><
                    +--,-showOpts--+
Generic form:

>>--moveTo(--newPos--+--------------+--)---------><
                     +--,-showOpts--+
```

31.18.38. redraw

```
WindowBase::redraw


>>--redraw---------------------------------------><
```

31.18.39. redrawClient

```
WindowBase::redrawClient


>>--redrawClient(--+------------+--)-------------><
                   +--eraseBkg--+
```

31.18.40. resize

```
WindowBase::resize
```

```
>>--resize(--width--,--height--+-------------+--)---------------><
                               +-,-showOpts--+
```

31.18.41. resizeTo

```
WindowBase::resizeTo


Form 1:

>>--resizeTo(--size--)---------------------------><

Form 2:

>>--resizeTo(--cx,--cy--)------------------------><

Generic form:

>>--resizeTo(--newSize--)------------------------><
```

31.18.42. screen2client

```
WindowBase::screen2client


>>--screen2client(--pointOrRect--)---------------><
```

31.18.43. screenToClient

```
WindowBase::screenToClient


>>--screenToClient(--x--,--y--)------------------><
```

31.18.44. sendMessage

```
WindowBase::sendMessage


>>--sendMessage(--id--,--msg--,--wParam--,--lParam--)-----------><
```

31.18.45. sendMessageHandle

```
sendMessageHandle


>>--sendMessageHandle(--id--,--msg--,--wParam--,--lParam--)-----><
```

31.18.46. setRect

```
WindowBase::setRect
```

```
Form 1:

>>--setRect(--rectangle--+------------+--)-------><
                         +-,-showOpts-+

Form 2:

>>--setRect(--point--,--size--+------------+--)-----------------><
                              +-,-showOpts-+

Form 3:

>>--setRect(--x-,--y-,--cx-,--cy--+------------+--)-------------><
                                  +-,-showOpts-+

Generic form:

>>--setRect(--ptSizeRectangle--+------------+--)----------------><
                               +-,-showOpts-+
```

31.18.47. setStyleRaw

```
WindowBase::setStyleRaw


>>--setStyleRaw(--value--)-----------------------><
```

31.18.48. setText

```
WindowBase::setText


>>--setText(--newText--)-------------------------><
```

31.18.49. setTitle

```
WindowBase::setTitle


>>--setTitle(--newText--)------------------------><
```

31.18.50. showFast

```
WindowBase::showFast


>>--showFast-------------------------------------><
```

31.18.51. title

```
WindowBase::title
```

```
>>--title----------------------------------------><
```

31.18.52. title=

```
WindowBase::title=


>>--title=newText--------------------------------><
```

31.18.53. update

```
WindowBase::update
```

31.18.54. windowRect

```
WindowBase::windowRect


>>--windowRect(--+--------+--)-------------------><
                 +--hwnd--+
```

31.19. WinTimer Routine

```
>>--WinTimer(--mode--,--msOrID--)----------------><
```

Creates, waits on, or releases, depending on *mode*, a periodic Windows timer.

Arguments:

The arguments are:

mode [required]

Exactly one of the following keywords, case is not significant. The *mode* keyword controls what the function does:

START	WAIT	STOP

START

Creates the timer and starts it. The WAIT and STOP modes will fail if the timer is not created first. In the START mode, the *msOrID* argument specifies the period, in milliseconds, of the timer. The return value for the START mode is the ID of the timer. This ID must then be used for the WAIT or STOP modes.

Waits until the next period of the timer is elapsed. In the WAIT mode, the *msOrID* argument must be the ID of the timer that is returned from using the CREATE mode. The return value is 0 on success and the system error code on error.

STOP

Stops, releases, the timer. In the STOP mode, the *msOrID* argument must be the ID of the timer that is returned from using the CREATE mode. The return value is 0 on success and the system error code on error.

msOrID [required]

The value of the *msOrID* argument is dependent on the *mode* argument as explained above. When creating the timer this argument is the desired period, in milliseconds, of the timer. In the WAIT and STOP modes, the *msOrID* argument must be the timer ID that is returned from calling the function in the CREATE mode.

Return value:

The return value is dependent on the *mode* the function is called with. If *mode* is CREATE, then the return value is 0 on error, otherwise the timer ID on success. In the other 2 modes the return is 0 on success and a system error code on error.

Remarks:

Every created timer should be released by calling the function with the STOP *mode*. Otherwise, the operating system will continue signaling the timer and consuming some system resources.

When calling *WinTimer* in the WAIT mode, the routine does not return until the current period of the timer elapses. During the WAIT, the current thread is suspended. When starting the WAIT, some portion of the current period will have elapsed, so the current thread is not suspended for the timer period. It is only suspended for the remaining time in the current period. This is unlike the Rexx *SysSleep* routine, where the thread is suspended for the interval specified. For instance, if the timer period is 2 seconds and the application calls the routine in WAIT mode after .9 seconds of the current period has elapsed, the thread will only be suspended for 1.1 seconds.

Windows is not a real time operating system. Any type of timer in Windows can only achieve an approximate wait period. In particular you can not expect a working wait period of less than 15 milliseconds with the *WinTimer* routine.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example that shows how to correctly start and use a timer with the *WinTimer* routine:

```
::method doStuff unguarded
  expose finished

  finished = .false
  reply 0

  id = WinTimer("START", 2000)
  if id <> 0 then do while \ finished
    say "Still working at:" time()

    do i = 1 to 10
      ...
    end

    ...

    ret = WinTimer("WAIT", id)
    if ret <> 0 then leave
  end

  if id <> 0 then ret = WinTimer("STOP", id)

  finished = .true
```

Windows Shell Objects

The ooDialog framework provides a number of classes that are used to access the Windows Shell. The Windows Shell objects are implemented using COM (the Component Object Model, a standard introduced by Microsoft.) The Windows Shell should not be confused with the Windows window manager, which is responsible for the windowing system in Windows. ooDialog is primarily concerned with, and mostly deals with, the windowing system. However, in a modern Windows system, not all tasks can be done through the windowing system. Selecting or choosing a folder is an example of a task that is not done through the window manager.

32.1. Shell Object Classes Table

The ooDialog classes and routines for working with shell objects are listed in the following table:

Table 32.1. The ooDialog Shell Object Classes

Class	Description
BrowseForFolder class	Provides access to the Windows *Browse For Folder* dialog, allowing complete configuration of the dialog.
ComConstants class	A *mixin* class containing constant values useful when working with shell objects.
CommonDialogCustomizations class	**Vista** or later only. A *mixin* class that provides a way to customize a **`CommonItemDialog`** object.
CommonItemDialog class	**Vista** or later only. The common item dialog was formerly known as the common file dialog. It is used in two variations, the open file dialog and the save file dialog.
CommonDialogEvents class	**Vista** or later only. Contains event handler methods that allow the ooDialog program to be notified of events within a common file dialog.
OpenFileDialog class	**Vista** or later only. A concrete subclass of the **`CommonItemDialog`** that displays an open file dialog to the user.
SaveFileDialog class	**Vista** or later only. A concrete subclass of the **`CommonItemDialog`** that displays a save file dialog to the user.
ShellItemFilter class	**Vista** or later only. Can be used if an application needs to perform special filtering to remove some items from the common item dialog's view.
SimpleFolderBrowse class	Allows the display of a simple to use, but non-configurable, Windows *Browse For Folder* dialog.
SimpleFolderBrowse routine	A short cut routine for the **`SimpleFolderBrowse`** dialog.

32.2. Shell / COM considerations

In general, the Rexx programmer can work with the *shell* object classes in the same way as working with any other ooDialog dialog classes. However, because the underlying Windows objects are COM objects rather than the usual windowing system window, there are a few considerations the programmer needs to be aware of.

COM Library / Threading:

The use of COM objects requirers that the COM library be initialized and released for *each* thread that a COM object is used in. Normally the Rexx programmer does not need to be much aware of threading and thread issues. However the COM requirement that the library be initialized and released on each thread changes that. Using the *BrowseForFolder* object as an example, the normal usage pattern would be:

- *Instantiate a **`BrowseForFolder`** object.*
- Configure the browse for folder object.
- Show the dialog and get the user's selection.

This usage pattern is the same for other shell objects such as a *OpenFileDialog* object, a *SaveFileDialog* object, etc..

As long as this is all done on one thread, there is no problem. The ooDialog framework initializes COM when the object is instantiated and releases COM when the underlying dialog is closed, that is when the *getFolder*, *getItemIDList*, *getResult*, etc., methods return. However, if the programmer wants, or needs, to implement a different usage pattern, then it becomes the programmer's responsibility to ensure that COM is initialized and released properly on each thread. The **`BrowseForFolder`** class, and other ooDialog shell classes, provide the tools the programmer needs to do that. If COM is not initialized on a thread the browse for folder object is running on, then methods invoked on the object may fail. If COM is not released on a thread it *was* initialized on, then system resource leaks can occur.

This is not that complicated and is not really that different than other areas of ooDialog concerning system resources. If the programmer wants to use a different font for a dialog control, the *setFont* method will fail unless the programmer has first initialized a font. If the programmer initializes a font using the *createFontEx* method and does not release the font, then system resource leaks can occur. The only difference here is that this *initialize / release* cycle is on a per thread basis.

The documentation for the individual attributes and methods of the shell object classes will make note of when the programmer needs to consider these threading issues when using the attribute or method. In addition, some of the ooDialog shell classes, such as the *SimpleFolderBrowse* class, provide simplified interfaces where the Rexx programmer does not need to be aware of COM.

Item ID List

The Windows Shell identifies objects in the Shell using a device called an item ID list. Files, folders, printers, remote computers all have an item ID list that is used to uniquely identify them to the Shell. The ooDialog programmer does not really need to know about the details of these item ID lists. However, it is difficult to document the shell classes without making references to an item ID list. Although technically incorrect, the programmer can just think of them as *handles*.

Shell Item Get Display Name

Each object in the Windows Shell is usually called a *shell item* in the MSDN *documentation*. As discussed above each object is uniquely identified by its item ID list. However, these IDs are not suitable for identifying a shell item to a Windows user. The shell provides a method for the programmer to get a suitable display name for every shell item. For any given shell item, the name of the item can be formatted in a number of different ways. The shell uses a special value called a *SIGDN* to specify the type of display name. In ooDialog, each SIGDN value has a keyword that the programmer uses to specify the display name type in methods that allow specifying the type. These key words are case insensitive. The keywords are:

DESKTOPABSOLUTEEDITING	PARENTRELATIVEFORADDRESSBAR
DESKTOPABSOLUTEPARSING	PARENTRELATIVEPARSING
FILESYSPATH	URL
PARENTRELATIVE	NORMALDISPLAY

PARENTRELATIVEEDITING

DESKTOPABSOLUTEEDITING
: Returns the editing name relative to the desktop.

DESKTOPABSOLUTEPARSING
: Returns the parsing name relative to the desktop.

FILESYSPATH
: Returns the display name relative to the file system path

PARENTRELATIVE
: MSDN does not document this.

PARENTRELATIVEEDITING
: Returns the editing name relative to the parent folder.

PARENTRELATIVEFORADDRESSBAR
: Returns the path relative to the parent folder in a friendly format as displayed in an address bar.

PARENTRELATIVEPARSING
: Returns the parsing name relative to the parent folder.

URL
: Returns the display name relative to a URL.

NORMALDISPLAY
: Returns the display name relative to the desktop.

COM result codes

In the COM API, methods invoked on COM objects return a result code called a HRESULT. This is just a number, very similar to the *system error code* of the Win32 API. However, there is no easy way to look up the meaning of a HRESULT code in the Windows *documentation* as there is with a system error code. The return from many, if not most, of the ooDialog shell object methods is the HRESULT code returned by the underlying COM objects. In a addition, all the ooDialog shell object methods set the *.SystemErrorCode* variable to the HRESULT code.

The *ComConstants* class provides some constants for common HRESULT codes. The **`S_OK`** constant is the HRESULT code for no error. Its value is 0 and the programmer can check for 0 or non-zero to test for success or failure of methods that return a HRESULT code.

HRESULT codes are often shown in hexadecimal format. The ooDialog shell object methods return HRESULT codes as a base 10 number. If the programmer converts the number to hexadecimal format, an Internet search will usually turn up the meaning of the error code. For example, one of the shell object methods could return 2147500068. Converted to hexadecimal this is 0x80004024. A search for 0x80004024 quickly leads to 0x80004024: The specified activation could not occur in the client context as specified. In addition, the *errMsg* method of the **`DlgUtil`** class will return a formatted error message string for any HRESULT code in either decimal or hexadecimal. The *errMsg* message works for system error codes also. For example:

```
hResult = 2147500068
sysErr  = 2

say 'HRESULT error code:'
say .DlgUtil~errMsg(hResult)
say
```

```
  say 'System error code:'
  say .DlgUtil~errMsg(sysErr)

::requires 'ooDialog.cls'

/* Output would be:
HRESULT error code:
Error code 2147500068 (0x80004024): The specified activation could not occur in the
 client context as specified.


System error code:
Error code 2 (0x00000002): The system cannot find the file specified.

HRESULT success code:
Error code 0 (0x00000000): The operation completed successfully.
*/
```

Note that the error message itself comes from Microsoft, ooDialog simply reports it and is not responsible for the clarity, or lack of clarity, of the message.

32.3. Special Folder Names

Windows uses CSIDL (constant special item ID list) names to provide a unique system-independent way to identify special folders used frequently by applications. These folders may not have the same name or location on any given system. For example, the system folder may be *C:\Windows* on one system and *C:\Winnt* on another.

In Windows Vista and later the CSIDL values have been replaced by Known Folder values, but for compatibility the CSIDL values still work. ooDialog does not yet support the known folder values, only the CSIDL names. Some of the methods of the *BrowseForFolder* class and the *SimpleFolderBrowse* class accept the CSIDL names listed below for arguments. Future enhancements to ooDialog may add other Shell classes that also accept the CSIDL names. The names are listed here in one place. The names are not case sensitive, but are otherwise spelled exactly as Microsoft does. Google can be used to research any of the names further. Only the names listed are recognized in ooDialog.

CSIDL_ADMINTOOLS:
: The file system directory that is used to store administrative tools for an individual user.

CSIDL_ALTSTARTUP:
: The file system directory that corresponds to the user's nonlocalized Startup program group. This value is recognized in Windows Vista for backward compatibility, but the folder itself no longer exists.

CSIDL_APPDATA:
: The file system directory that serves as a common repository for application-specific data.

CSIDL_BITBUCKET:
: The virtual folder that contains the objects in the user's Recycle Bin.

CSIDL_CDBURN_AREA:
: The file system directory that acts as a staging area for files waiting to be written to a CD.

CSIDL_COMMON_ADMINTOOLS:
: The file system directory that contains administrative tools for all users of the computer.

CSIDL_COMMON_ALTSTARTUP:
: The file system directory that corresponds to the nonlocalized Startup program group for all users. Valid only for Microsoft Windows NT systems. This value is recognized in Windows Vista for backward compatibility, but the folder itself no longer exists.

CSIDL_COMMON_APPDATA:
: The file system directory that contains application data for all users. This folder is used for application data that is not user specific. For example, an application can store a spell-check dictionary, a database of clip art, or a log file in the CSIDL_COMMON_APPDATA folder. This information will not roam and is available to anyone using the computer.

CSIDL_COMMON_DESKTOPDIRECTORY:
: The file system directory that contains files and folders that appear on the desktop for all users.

CSIDL_COMMON_DOCUMENTS:
: The file system directory that contains documents that are common to all users.

CSIDL_COMMON_FAVORITES:
: The file system directory that serves as a common repository for favorite items common to all users

CSIDL_COMMON_MUSIC:
: The file system directory that serves as a repository for music files common to all users.

CSIDL_COMMON_OEM_LINKS:
: This value is recognized in Windows Vista for backward compatibility, but the folder itself is no longer used.

CSIDL_COMMON_PICTURES:
: The file system directory that serves as a repository for image files common to all users

CSIDL_COMMON_PROGRAMS:
: The file system directory that contains the directories for the common program groups that appear on the Start menu for all users.

CSIDL_COMMON_STARTMENU:
: The file system directory that contains the programs and folders that appear on the Start menu for all users

CSIDL_COMMON_STARTUP:
: The file system directory that contains the programs that appear in the Startup folder for all users.

CSIDL_COMMON_TEMPLATES:
: The file system directory that contains the templates that are available to all users.

CSIDL_COMMON_VIDEO:
: The file system directory that serves as a repository for video files common to all users.

CSIDL_COMPUTERSNEARME:
: The folder that represents other computers in your workgroup.

CSIDL_CONNECTIONS:
: The virtual folder that represents Network Connections, that contains network and dial-up connections.

CSIDL_CONTROLS:
: The virtual folder that contains icons for the Control Panel applications.

CSIDL_COOKIES:
: The file system directory that serves as a common repository for Internet cookies.

CSIDL_DESKTOP:
: The virtual folder that represents the Windows desktop, the root of the namespace.

CSIDL_DESKTOPDIRECTORY:
: The file system directory used to physically store file objects on the desktop (not to be confused with the desktop folder itself.)

CSIDL_DRIVES:
: The virtual folder that represents My Computer, containing everything on the local computer: storage devices, printers, and Control Panel.

CSIDL_FAVORITES:
: The file system directory that serves as a common repository for the user's favorite items.

CSIDL_FONTS:
: A virtual folder that contains fonts.

CSIDL_HISTORY:
: The file system directory that serves as a common repository for Internet history items.

CSIDL_INTERNET:
: A virtual folder for Internet Explorer.

CSIDL_INTERNET_CACHE:
: The file system directory that serves as a common repository for temporary Internet files

CSIDL_LOCAL_APPDATA:
: The file system directory that serves as a data repository for local (nonroaming) applications.

CSIDL_MYDOCUMENTS:
: The virtual folder that represents the My Documents desktop item.

CSIDL_MYMUSIC:
: The file system directory that serves as a common repository for music files.

CSIDL_MYPICTURES:
: The file system directory that serves as a common repository for image files.

CSIDL_MYVIDEO:
: The file system directory that serves as a common repository for video files

CSIDL_NETHOOD:
: A file system directory that contains the link objects that may exist in the My Network Places virtual folder. It is not the same as CSIDL_NETWORK, which represents the network namespace root.

CSIDL_NETWORK:
: A virtual folder that represents Network Neighborhood, the root of the network namespace hierarchy.

CSIDL_PERSONAL:
: The virtual folder that represents the My Documents desktop item.

CSIDL_PRINTERS:
: The virtual folder that contains installed printers.

CSIDL_PRINTHOOD:
: The file system directory that contains the link objects that can exist in the Printers virtual folder.

CSIDL_PROFILE:
: The user's profile folder. Applications should not create files or folders at this level; they should put their data under the locations referred to by CSIDL_APPDATA or CSIDL_LOCAL_APPDATA.

CSIDL_PROGRAM_FILES:
: The Program Files folder.

CSIDL_PROGRAM_FILES_COMMON:
: A folder for components that are shared across applications.

CSIDL_PROGRAM_FILES_COMMONX86:
: Listed in the MSDN documentation with no explanation.

CSIDL_PROGRAM_FILESX86:
: Listed in the MSDN documentation with no explanation.

CSIDL_PROGRAMS:
: The file system directory that contains the user's program groups (which are themselves file system directories.)

CSIDL_RECENT:
: The file system directory that contains shortcuts to the user's most recently used documents.

CSIDL_RESOURCES:
: Windows Vista. The file system directory that contains resource data.

CSIDL_RESOURCES_LOCALIZED:
: Listed in the MSDN documentation with no explanation.

CSIDL_SENDTO:
: The file system directory that contains Send To menu items.

CSIDL_STARTMENU:
: The file system directory that contains Start menu items.

CSIDL_STARTUP:
: The file system directory that corresponds to the user's Startup program group. The system starts these programs whenever any user logs onto Windows.

CSIDL_SYSTEM:
: The Windows System folder.

CSIDL_SYSTEMX86:
: Listed in the MSDN documentation with no explanation.

CSIDL_TEMPLATES:
: The file system directory that serves as a common repository for document templates

CSIDL_WINDOWS:
: The Windows directory or SYSROOT. This corresponds to the %windir% or %SYSTEMROOT% environment variables.

32.4. BrowseForFolder Class

A BrowseForFolder object represents the Windows *Browse For Folder* common dialog. The Windows operating system provides a number of common dialogs for use by applications to allow presenting the user with an unified generic method to do common tasks. The purpose of the Browse For Folder dialog is specifically to allow the user to open or choose folders.

32.4.1. Method Table

The following table lists the class and instance methods of the **`BrowseForFolder`** class:

Table 32.2. BrowseForFolder Class Method Reference

Method	Description
Class Methods	**Class Methods**
new	Instantiates a new **`BrowseForFolder`** object
Attribute Methods	**Attribute Methods**
banner	Reflects the banner text for this browse for folder object.
dlgTitle	Reflects the title to be used in the browse for folder dialog.
hint	Reflects the hint text to be used in the browse for folder dialog
initialThread	Reflects the thread ID of the thread this browse for folder object was instantiated on.
options	Reflects the Browse For Folder dialog options set for this object.
owner	Reflects the Rexx dialog that is the owner of this browse for folder dialog. By default there is no owner.
root	Reflects the folder that will be set as the root folder of the tree-view control in the browse for folder dialog.
startDir	Reflects a directory, a folder, that should be pre-selected when the browse for folder dialog first opens.
usePathForHint	Reflects the status of the internal flag controlling whether the hint text should be set to the selected folder's path.
Instance Methods	**Instance Methods**
getDisplayName	Returns the display name for an item *ID* list.
getFolder	Shows the browse for folder dialog configured as defined by the current values of the attributes of this object.
getItemIDList	Shows the browse for folder dialog and retrieves the item *ID* list the user picks.
initCOM	Initializes the COM *library* on the current thread.
releaseCOM	Releases the COM *library* on the current thread.
releaseItemIDList	

32.4.2. new (Class Method)

```
>>--new(--+---------+--+----------+--+--------+--+-----------+--)------------><
          +--title--+  +-,-banner-+  +-,-hint-+  +-,-startDir-+
```

Instantiates a new **`BroweForFolder`** object.

Arguments:

The arguments are:

title [optional]

Specifies a caption, or title, for the dialog. The Windows default title is *Browse For Folder*. If this argument is used, the *dlgTitle* attribute is set using the value specified.

If this argument is omitted, the default ooDialog value, *ooDialog Browse for Folder*, is set for the *dlgTitle* attribute. To prevent the ooDialog default value from being used and have the Windows default used, set this argument to the **`.nil`** object, or to the empty string.

banner [optional]

Specifies text to use for what Windows calls the *banner* for the dialog. This string is displayed above the tree view control in the dialog box. The string can be used to specify instructions to the user. By default, Windows, does not display the banner and the area above the tree view is blank. If this argument is used, the *banner* attribute is set to the value specified.

If this argument is omitted, the default ooDialog value, *Select the folder needed*, is set for the *banner* attribute. To prevent the ooDialog default value from being used and have the Windows default used, set this argument to the **`.nil`** object, or to the empty string.

hint [optional]

Specifies text to use for a *hint* in the dialog. The hint is displayed below the bottom of the tree view control and above the bottom row of buttons of the dialog. By default, Windows does not display a hint, and the area where the hint would go is removed from the dialog. That is, the bottom row of buttons is placed directly below the tree view. If this argument is used, the *hint* attribute is set to the value specified.

If this argument is omitted, the default ooDialog value, *If the needed folder does not exist it can be created*, is set for the *hint* attribute. To prevent the ooDialog default value from being used and have the Windows default used, set this argument to the **`.nil`** object, or to the empty string.

startDir [optional]

Specifies an initial selected folder for the dialog. By default, Windows opens the dialog with the top-most item in the tree view selected. If a start folder is specified, the folder is selected and, if needed, the tree view items are expanded and the tree view is scrolled so that the starting folder is visible in the tree view. If this argument is used, the *startDir* attribute is set to the value specified.

Testing indicates that the folder specified must be a fully qualified path name of an existing directory. If not, the operating system seems to just ignore it.

If this argument is omitted, the ooDialog framework sets the *startDir* attribute to the empty string. The causes Windows to use its default behaviour.

Return value:

Returns a newly instantiated **`BrowseForFolder`** object.

Remarks:

During instantiation, the *options*, *root*. and the *initialThread* attributes are also set.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example instantiates a new **`BrowseForFolder`** object with a title, banner, and hint:

```
-- Set title, banner, and a hint for the dialog.
title  = 'Browse For a Printer'
banner = 'Select the printer to use for this test.'
hint   = 'This is only a demonstration, no printing will be done.'

bff = .BrowseForFolder~new(title, banner, hint)
```

32.4.3. banner (Attribute)

```
>>--banner---------------------------------------><

>>--banner = text--------------------------------><
```

Reflects the banner text for this browse for folder object. The *banner* is an area above the tree-view control in the dialog that can be used to display a few lines of text. By default, Windows leaves this area blank.

banner get:

Returns the banner text for this object, or the empty string if the Windows default behaviour for the banner is to be used.

banner set:

To use a non-default banner, set this attribute to the text desired. To specify that the default Windows behaviour regarding the banner be used, set this attribute to the empty string or to the **`.nil`** object.

Remarks:

By default, the text, *Select the folder needed* is set for the banner by the ooDialog framework. By default, Windows will not use a banner. To use the Windows default behaviour rather than the ooDialog default banner, set the *banner* attributed to the empty string, or the **`.nil`** object.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example example configures a browse for folder object to not display a banner:

```
bff = .BrowseForFolder~new('Choose the Backup Folder')
bff~banner = .nil
```

32.4.4. dlgTitle (Attribute)

```
>>--dlgTitle-------------------------------------><
```

```
>>--dlgTitle = title-----------------------------><
```

Reflects the title to be used in the browse for folder dialog.

dlgTitle get:

Returns the text to be used for the dialog title, or the empty string if the Windows default title will be used.

dlgTitle set:

Set this attribute to the title desired for the dialog, if the ooDialog default is not desired. Set this attribute to the empty string or to the **`.nil`** object to specify that the Windows default title be used.

Remarks:

By default the ooDialog framework sets the *dlgTitle* attribute to *ooDialog Browse for Folder*. The Windows default title is *Browse fo Folder.* To have the dialog use the Windows default, set this attribute to the empty string or the **`.nil`** object.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example instantiates a browse for folder dialog and then sets a custom title for the dialog:

```
  bff = .BrowseForFolder~new
  bff~dlgTitle ='Choose the Backup Folder'
```

32.4.5. hint (Attribute)

```
>>--hint-----------------------------------------><

>>--hint = hint----------------------------------><
```

Reflects the hint text to be used in the browse for folder dialog. The *hint* is an area between the bottom of the tree-view and above the bottom rows of buttons that can be used to display a line of text. By default, Windows removes this area from the dialog.

hint get:

Returns the text of the hint for this browse for folder dialog. Returns the empty string if no hint is to be displayed.

hint set:

Set this attribute to the text to display in the hint area, if the default ooDialog hint is not appropriate. Set this attribute to the empty string or the **`.nil`** object to restore the default Windows behaviour, which is to remove the hint area altogether.

Remarks:

By default, the ooDialog framework sets this attribute to *If the needed folder does not exist it can be created*. To prevent this text being used, either set the *hint* attriubte to the text desired, or set it to the empty string to remove the hint area altogether.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example instantiates a browse for folder object and then sets the hint to text that informs the user what to do:

```
bff = .BrowseForFolder~new
bff~hint ='Choose the folder to place your back up files in.'
```

32.4.6. initialThread (Attribute)

```
>>--initialThread--------------------------------><
```

Reflects the thread ID of the thread this browse for folder object was instantiated on.

initialThread get:

Returns the thread ID for the thread that this object was instantiated on.

initialThread set:

The *initialThread* attribute can not be set by the programmer. It is set by the ooDialog framework and can not be changed.

Remarks:

When the browse for folder dialog is first instantiated, the ooDialog framework *initializes* the COM library on the thread that is executing and records the thread ID in the *initialThread* attribute. If the application makes use of the browse for folder object in multiple threads, the value of the initial thread ID is helpful in determining if the COM library needs to be initialized or not on the currently executing thread. The *threadID* method of the *DlgUtil* class can be used to determine the ID of the currently executing.

The recommendation is that the browse for folder dialog should be instantiated, used, and disposed of all on the same thread. However, if the programmer has a need, or even just the desire, to use the browse for folder object on multiple threads, there is no reason why that can not be done. The programmer just needs to manage the initialization and release of the COM library herself.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows a browse for folder object that is reused for the life-time of the dialog. In the onBrowse() method, if the method is not running on the same thread as the BrowseForFolder object was instantiated on, then COM is initialized and released each time the object is used. If it is the same thread, COM is not initialized and not released.

Finally, during the leaving() method, if the method is running on the same thread as the browse for folder was instantiated on, the item *ID* list and COM are released.

Note that this example does run correctly. But, if it were to turn out that the leaving() method was not running on the same thread as the define() method had run on, then COM and the item ID list would not be released correctly. It was only after some testing that it was determined that the program ran correctly:

```
::method defineDialog
  expose bff pidl
  ...

  say 'defineDialog() thread ID' .DlgUtil~threadID
  say
  bff = .BrowseForFolder~new
  bff~usePathForHint = .true
  bff~hint = ''

  pidl = .nil

::method onBrowse unguarded
  expose bff pidl

  say 'onBrowse() thread:' .DlgUtil~threadID
  say

  newThread = (bff~initialThread == .DlgUtil~threadID)

  if newThread then bff~initCOM

  if pidl \== .nil then bff~releaseItemIDList(pidl)
  if bff~owner \== .nil then bff~owner = self

  pidl = bff~getItemIDList(.true)

  if pidl \== .nil then say 'The user picked:' bff~getDisplayName(pidl)
  else say 'The user canceled'
  say

  if newThread then bff~releaseCOM

  return 0

::method leaving unguarded
  expose bff pidl

  if bff~initialThread == .DlgUtil~threadID then do
      say 'leaving() thread ID' .DlgUtil~threadID
      if pidl \== .nil then bff~releaseItemIDList(pidl)
      bff~releaseCOM
  end

/* Output from a typical session:

defineDialog() thread ID 4032

onBrowse() thread: 3320

The user picked: C:\Rexx\ooRexx.3.0.0.release

onBrowse() thread: 3320

The user canceled

onBrowse() thread: 3320

The user picked: U:\PictureFiles

leaving() thread ID 4032

*/
```

The recommendation is to simply instantiate a browse for folder object, use it, and dispose of it. The above example could have just as easily, and perhaps more easily, been written this way:

```
::method defineDialog
```

```
  ...

  say 'defineDialog() thread ID' .DlgUtil~threadID
  say
  return 0


::method onBrowse unguarded

  say 'onBrowse() thread:' .DlgUtil~threadID
  say

  bff = .BrowseForFolder~new
  bff~usePathForHint = .true
  bff~hint = ''
  bff~owner = self

  pidl = bff~getItemIDList(.true)

  if pidl \== .nil then say 'The user picked:' bff~getDisplayName(pidl)
  else say 'The user canceled'
  say

  if pidl \== .nil then bff~releaseItemIDList(pidl)
  bff~releaseCOM

  return 0

::method leaving unguarded

  say 'leaving() thread ID' .DlgUtil~threadID
  return 0

/* Output from a typical session:

defineDialog() thread ID 3580

onBrowse() thread: 3960

The user picked: C:\Rexx\ooRexx.3.2.0.debug

onBrowse() thread: 3960

The user picked: U:\MovieFiles\2013\Santa Fe New Mexico

onBrowse() thread: 3960

The user canceled

onBrowse() thread: 3960

The user picked: C:\Tools\TortoiseSVN

leaving() thread ID 3580

*/
```

32.4.7. options (Attribute)

```
>>--options--------------------------------------><

>>--options = keywords---------------------------><
```

Reflects the Browse For Folder dialog options set for this object.

options get:

Returns a list of the keywords that specify the Browse For Folder dialog options that are currently set for this object. The keywords are listed and explained in the remarks section.

options set:

Set the Browse For Folder dialog options using a blank separated list of keywords. The list is case insensitive. The possible keywords are listed in the remarks section.

Remarks:

On instantiation, the Browse For Folder dialog options are set to a default value that is appropriate for most use cases. The default options are *RETURNONLYFSDIRS RETURNFSANCESTORS NEWDIALOGSTYLE*. To change these options after instantiations, set this attribute using a list of 0 or more of the following keywords, case is not significant. The list *replaces* the current list, with one caveat. The ooDialog framework manages the NEWDIALOGSTYLE and UAHINT options. The NEWDIALOGSTYLE is required. If the new list does not contain that keyword it is added. The UAHINT option is added when needed, and removed when not needed.

BROWSEFILEJUNCTIONS	DONTGOBELOWDOMAIN	RETURNONLYFSDIRS
BROWSEFORCOMPUTER	NEWDIALOGSTYLE	SHAREABLE
BROWSEFORPRINTER	NONEWFOLDERBUTTON	UAHINT
BROWSEINCLUDEFILES	NOTRANSLATETARGETS	
BROWSEINCLUDEURLS	RETURNFSANCESTORS	

BROWSEFILEJUNCTIONS

Allows folder junctions like zip files and libraries to be browsed.

BROWSEFORCOMPUTER

Only return computers.

BROWSEFORPRINTER

Only allow the selection of printers.

BROWSEINCLUDEFILES

The browse dialog box will display files as well as folders.

BROWSEINCLUDEURLS

The browse dialog box can display URLs. The NEWDIALOGSTYLE and BROWSEINCLUDEFILES flags must also be set. If these three flags are not set, the browser dialog box will reject URLs. Even when these flags are set, the browse dialog box will only display URLs if the folder that contains the selected item supports them.

DONTGOBELOWDOMAIN

Do not include network folders below the domain level in the dialog box's tree view control.

NEWDIALOGSTYLE

Use the new user interface. Setting this flag provides the user with a larger dialog box that can be resized. The dialog box has several new capabilities, including: drag-and-drop capability within the dialog box, reordering, shortcut menus, new folders, delete, and other shortcut menu commands. The programmer can not effectively remove this flag, the ooDialog framework ensures it is always set.

NONEWFOLDERBUTTON

Do not include the New Folder button in the browse dialog box.

NOTRANSLATETARGETS
When the selected item is a shortcut, return the *item ID list* of the shortcut itself rather than its target.

RETURNFSANCESTORS
Only return file system ancestors. An ancestor is a subfolder that is beneath the root folder in the namespace hierarchy.

RETURNONLYFSDIRS
Only return file system directories.

SHAREABLE
The browse dialog box can display shareable resources on remote systems. It is intended for applications that want to expose remote shares on a local system.

UAHINT
Adds a usage hint to the dialog box. This option is set by the ooDialog framework, the programmer can not effectively alter it. The option is set if the *hint* attribute is set and removed if the attribute is not set.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example comes from a demonstration program showing how to browse for printers. After instantiating the browse for folder object, it sets the dialog options to a non-default value, a value appropriate for printer browsing:

```
title  = 'Browse For a Printer'
banner = 'Select the printer to use for this test.'
hint   = 'This is only a demonstration, no printing will be done.'

bff = .BrowseForFolder~new(title, banner, hint)

bff~owner = self
bff~root = 'CSIDL_PRINTERS'

bff~options = 'BROWSEFORPRINTER NONEWFOLDERBUTTON'
```

32.4.8. owner (Attribute)

```
>>--owner----------------------------------------><

>>--owner = ownerDlg-----------------------------><
```

Reflects the Rexx dialog that is the owner of this browse for folder dialog. By default there is no owner.

owner get:

If this browse for folder dialog has an owner dialog, returns that Rexx dialog. Returns the **`.nil`** object if ther is no owner dialog.

owner set:

To assign an owner dialog to the browse for folder dialog set the *ownerDlg* attribute to a Rexx dialog object that is to be the owner. By default there is no owner dialog assigned. If the programmer has set an owner dialog and for some reasons needs to remove it, this can be done by setting the *ownerDlg* attribute to the **.nil** object.

Remarks:

If the browse for folder dialog has an owner dialog, then when the browse dialog is displayed the operating system will disable the owner dialog until the user has closed the browse dialog. This is normally the behaviour wanted.

Note that setting the *owner* attribute before the underlying owner dialog has been created will have not effect.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows the first steps in configuring a *browse for printers* dialog that is displayed from a dialog. The dialog is set as the owner of the browse for printers dialog:

```
bff = .BrowseForFolder~new(title, banner, hint)
bff~owner = self
bff~root = 'CSIDL_PRINTERS'
...
```

32.4.9. root (Attribute)

```
>>--root-----------------------------------------><

>>--root = rootFolder----------------------------><
```

Reflects the folder that will be set as the root folder of the tree-view control in the browse for folder dialog. It is not possible for the user to browse above the root folder.

root get:

Returns the item *ID* list for the folder set as the root folder of the browse for folder dialog. If no root folder has been set, the **.nil** object is returned.

root set:

The *root* attribute can be set using a number of different forms. By default, no root folder is set. To remove a root folder, set the attribute to the empty string or to the **.nil** object. Use a full path name to set the root folder to the named folder. One of the *Special Folder Names* can be used. In addition, an item *ID* list can be used. An item ID list can be obtained through the *getItemIDList* method.

Remarks:

When an item ID list is obtained through the *getItemIDList*, the user is normally responsible for *releasing* the item ID list. However, the ooDialog automatically releases the item ID list set as the *root* attribute. There is no need to release an item ID list that has been set as the *root* attriubte.

Details:

Raises syntax errors when incorrect usage is detected.

Example:

This example comes from a demonstration program showing how to browse for printers. After instantiating the browse for folder object, it sets the root attribute to the printers folder:

```
title  = 'Browse For a Printer'
banner = 'Select the printer to use for this test.'
hint   = 'This is only a demonstration, no printing will be done.'

bff = .BrowseForFolder~new(title, banner, hint)

bff~owner = self
bff~root = 'CSIDL_PRINTERS'

bff~options = 'BROWSEFORPRINTER NONEWFOLDERBUTTON'
```

32.4.10. startDir (Attribute)

```
>>--startDir-------------------------------------><

>>--startDir = initialFolder---------------------><
```

Reflects a directory, a folder, that should be pre-selected when the browse for folder dialog first opens. The tree-view control is expanded and scrolled, if needed, so that the selected folder is visible when the dialog is shown.

startDir get:

Returns the full path name of the folder that is initially selected for this browse for folder dialog, of the empty string if no start folder has been set.

startDir set:

Set the *startDir* attribute to the full path name of the directory that should be pre-selected when the browse for folder dialog is started. To remove a starting directory, set this attribute to the **`.nil`** object or to the empty string.

Remarks:

Relative path names and incomplete path names are ignored by the operating system. The *startDir* attribute needs to be set to a full path name for this feature to work.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example configures the browse for folder dialog to open with the **`C:\Program Files \Microsoft Games`** directory selected:

```
bff = .BrowseForFolder~new

bff~root = 'CSIDL_DRIVES'
bff~usePathForHint = .true
bff~startDir = "C:\Program Files\Microsoft Games"

folder = bff~getFolder
```

32.4.11. usePathForHint (Attribute)

```
>>--usePathForHint-------------------------------><
```

```
>>--usePathForHint = trueOrFalse-----------------><
```

Reflects the status of the internal flag controlling whether the hint text should be set to the selected folder's path. By default the flag is set to false.

usePathForHint get:

Returns true if the use path for hint feature is enabled, or false if it is disabled.

usePathForHint set:

Set the *usePathForHint* attribute to true to enable the updating of the hint to the path of the currently selected folder, or to false to disable the feature.

Remarks:

The browse for folder dialog has a feature that will update the hint text with the full path to the currently selected folder in the tree-view control. Each time the selected folder is changed, the hint is updated with the new path to the newly selected folder. This feature is turned on or off throught the *usePathForHint* attribute.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example instantiates a browse for folder object and configures it to display the full path to the selected folder as the hint by setting the *usePathForHint* attribute to true:

```
bff = .BrowseForFolder~new

bff~root = 'CSIDL_MYDOCUMENTS'
bff~usePathForHint = .true

folder = bff~getFolder
```

32.4.12. getDisplayName

```
>>--getDisplayName(--pidl--,--+--------+--)------><
                              +--type--+
```

Returns the display name for an item *ID* list.

Arguments:

The arguments are:

pidl [required]

The item ID list whose display name is desired.

type [optional]

A *SIGDN* keyword that specfies the format of the returned display name. By default if this arugment is omitted, the method uses the FILESYSPATH keyword. But, if that fails, it tries again using the DESKTOPABSOLUTEEDITING keyword. This should always return something. When used, the *type* argument specifies the format of the returned display name using exactly one of the *SIGDN* keywords.

Return value:

Returns the display name on success, the **`.nil`** object on failure.

Remarks:

Not all format types will return a display name for any arbitrary item ID list. For example, if the item ID list is a virtual folder and the FILESYSPATH format is requested, the method will fail.

If the *type* argument is omitted this method should return some name for any shell item, virtual folder or not. In this case the method first tries to get a full file path name. But, if that fails the method tries again using the flag for desktop absolute editing. If the *type* argument is used, then the method uses the specified SIGDN flag as is and returns whatever the shell returns.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example gets an item ID list that the user selects and then prints its display name:

```
  bff = .BrowseForFolder~new
  bff~root = 'CSIDL_DRIVES'
  bff~usePathForHint = .true
  bff~startDir = "C:\Program Files\Microsoft Games"
  idList = bff~getItemIDList
  say bff~getDisplayName(idList, 'NORMALDISPLAY')
  return 0

/* Output might be:

$Recycle.Bin

*/
```

32.4.13. getFolder

```
>>--getFolder(--+---------+--)-------------------><
                +--reuse--+
```

Shows the browse for folder dialog configured as defined by the current values of the attributes of this object.

Arguments:

The single argument is:

reuse [optional]

True or false to specify whether this object will be *reused* or if the programmer is finished with the object. The default is false, the programmer is done invoking methods on the object

Return value:

Returns the folder selected by the user, or the `.nil` object if the user cancels the dialog.

Remarks:

The recommended usage pattern for the BrowseForFolder object is to instantiate the object, configure the dialog through the object's attributes, and show the dialog using *getFolder*, or *getItemIDList*. In this way the COM library is *initialized* when the object is instantiated and released during the return from *getFolder* or *getItemIDList*.

The COM library is always released during the *getFolder* method, unless the *reuse* argument is true. When *reuse* is true, the COM library is not released and it becomes the programmer's responsibility to ensure the library is released properly

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example instantiates the **`BrowseForFolder`** object, configures it, and shows it. On return it checks if the user canceled the dialog or not. Since the programmer is done with the object, the *reuse* argument is omitted:

```
bff = .BrowseForFolder~new
bff~usePathForHint = .true
bff~hint = ''
bff~owner = self

folder = bff~getFolder

if folder \== .nil then say 'The user picked:' folder
else say 'The user canceled'
```

32.4.14. getItemIDList

```
>>--getItemIDList(--+---------+--)---------------><
                    +--reuse--+
```

Shows the browse for folder dialog and retrieves the item *ID* list the user picks.

Arguments:

The single argument is:

reuse [optional]

True or false to specify whether this object will be *reused* or if the programmer is finished with the object. The default is false, the programmer is done invoking methods on the object

Return value:

Returns the itemID list the user picks, or the **`.nil`** object if the user cancels the dialog.

Remarks:

The item ID list returned is similar to a *handle* to other system resources, like a *font* or a *device context*, and must be released to free up the resource. Use the *releaseItemIDList* method to free the item ID list when done with it. **Note** that the *releaseItemIDList* does not need the COM library to be initialized. Therefore there is no problem invoking *releaseItemIDList after* the COM library has been released.

However, if the returned item ID list is set as the *root* attribute, there is no need to release that itme ID list, the ooDialog framework will release it automatically.

The recommended usage pattern for the BrowseForFolder object is to instantiate the object, configure the dialog through the object's attributes, and show the dialog using *getItemIDList*, or *getFolder*. In this way the COM library is *initialized* when the object is instantiated and released during the return from *getFolder* or *getItemIDList*.

The COM library is always released during the *getItemIDList* method, unless the *reuse* argument is true. When *reuse* is true, the COM library is not released and it becomes the programmer's responsibility to ensure the library is released properly

Details

Raises syntax errors when incorrect usage is detected.

Example:

In this example the user is first asked to pick the top-level folder for all backups the application does. Then, the user is asked to select the folder for the current backup. To allow the user to pick a virtual folder like the Documents subfolder in Libraries, the application uses the *getIDList* method for the first showing of the browse for folder dialog. The returned item ID list is then set as the root for the second showing of the browse for folder dialog. This ensures that the user picks a subfolder of the first folder picked.

Some other things to note about the example. For the first showing of the options attribute is set to the empty string and the root is set to CSIDL_DESKTOP. CSIDL_DESKTOP is the root of the entire shell namespace. This combination allows the browsing of every folder in the shell. In the first invocation of *getItemIDList*, note the use of the *reuse* arg. The prevents the COM library from being released. Then in the last invocation of *getFolder*, the *reuse* arg is not used, and the ooDialog framework takes care of releasing the COM library. However, if the user cancels the first showing of the dialog, the application handles releasing COM before it returns:

```
title = 'Acme Widgets - A1 Accounting Program'
banner = 'The A1 Accounting Program needs to know the top-level' || -
         ' folder for data backups'

bff = .BrowseForFolder~new
bff~dlgTitle = title
bff~banner = banner
bff~hint = 'Pick, or create, the folder for your backups.'
bff~options = '';
bff~root = 'CSIDL_DESKTOP'

idList = bff~getItemIDList(.true)
if idList == .nil then do
  say 'The user canceled'
  bff~releaseCOM
  return .nil
end

say 'Top-level backup folder:' bff~getDisplayName(idList, 'NORMALDISPLAY')

newBanner = 'The A1 Accounting Program needs to know the subfolder' || -
            ' for this backup.'
bff~root = idList
bff~options = bff~options 'RETURNFSANCESTORS RETURNONLYFSDIRS'
bff~banner = newBanner
bff~hint = 'Pick, or create, the folder for this backup.'
bff~startDir = .nil

folder = bff~getFolder

select
  when folder == .nil then say 'The user picked a virtual folder'
  when folder == '' then say 'The user canceled'
  otherwise say 'The user picked:' folder
end
-- End select

return folder
```

```
::requires 'ooDialog.cls'
```

32.4.15. initCOM

```
>>--initCOM--------------------------------------><
```

Initializes the COM *library* on the current thread.

Arguments:

There are no arguments for this method.

Return value:

Returns true when the COM library is initialized as a result of the *initCom* invocation. Returns false if the library is not initialized as a result of the *initCom* invocation.

Remarks:

When the browse for folder dialog is first instantiated, the ooDialog framework initializes the COM library on the thread that is executing. If that same object is used on another thread than the initial thread, the methods of the object may fail unless the COM library is initialized on the other thread. The *initCOM* method is provided to perform that initialization.

For each invocation of *initCOM* that returns true, a matching call to *releaseCOM* must be made on the same thread. Note that the *getFolder* and *getItemIDList* methods release the COM library on the thread they are executing in when the *reuse* arguemnt is false. Therefore, if *initCOM* is invoked on thread xx, the matching release COM requirement can be fulfilled by invoking *getFolder* with the *reuse* argument set to false.

If *initCOM* returns false, then there is no need to invoke *releaseCOM*. Indeed, invoking *releaseCOM* will be a mistake as this will cause the matching *initCOM* / *releaseCOM* calls to become unbalanced. When *initCom* returns false, it can be for two reasons. If the COM library has already been initialized on the current thread, false is returned and the **`.SystemErrorCode`** is set to 1. There is also the possibility of a failure in initializing the COM library. In this case, false is returned and the **`.SystemErrorCode`** is set to a COM error code that describes the error.

Details

Sets the *.SystemErrorCode* variable.

Example:

This example shows a browse for folder object that is reused for the life-time of the dialog. In the onBrowse() method, if the method is not running on the same thread as the BrowseForFolder object was instantiated on, then COM is initialized and released each time the object is used. If it is the same thread, COM is not initialized and not released.

Finally, during the leaving() method, if the method is running on the same thread as the browse for folder was instantiated on, the item *ID* list and COM are released.

Note that this example does run correctly. But, if it were to turn out that the leaving() method was not running on the same thread as the define() method had run on, then COM and the item ID list would not be released correctly. It was only after some testing that it was determined that the program ran correctly:

```
::method defineDialog
  expose bff pidl
  ...

  say 'defineDialog() thread ID' .DlgUtil~threadID
  say
  bff = .BrowseForFolder~new
  bff~usePathForHint = .true
  bff~hint = ''

  pidl = .nil

::method onBrowse unguarded
  expose bff pidl

  say 'onBrowse() thread:' .DlgUtil~threadID
  say

  newThread = (bff~initialThread == .DlgUtil~threadID)

  if newThread then bff~initCOM

  if pidl \== .nil then bff~releaseItemIDList(pidl)
  if bff~owner \== .nil then bff~owner = self

  pidl = bff~getItemIDList(.true)

  if pidl \== .nil then say 'The user picked:' bff~getDisplayName(pidl)
  else say 'The user canceled'
  say

  if newThread then bff~releaseCOM

  return 0

::method leaving unguarded
  expose bff pidl

  if bff~initialThread == .DlgUtil~threadID then do
      say 'leaving() thread ID' .DlgUtil~threadID
      if pidl \== .nil then bff~releaseItemIDList(pidl)
      bff~releaseCOM
  end

/* Output from a typical session:

defineDialog() thread ID 4032

onBrowse() thread: 3320

The user picked: C:\Rexx\ooRexx.3.0.0.release

onBrowse() thread: 3320

The user canceled

onBrowse() thread: 3320

The user picked: U:\PictureFiles

leaving() thread ID 4032

*/
```

32.4.16. releaseCOM

```
>>--releaseCOM-----------------------------------><
```

Releases the COM *library* on the current thread.

Arguments:

There are no arguments for this method.

Return value:

Returns true if the current thread is the same thread as this **`BrowseForFolder`** object was instantiated on, otherwise false. A return of false does not indicate failure.

Remarks:

As noted through out the **`BrowseForFolder`** documentation, the COM library must be initialized and released for each thread the browse for folder object is used on. The ooDialog initializes the COM library on the thread the object is first instantiated on and releases the COM library on the thread that the *getFolder* method, or the *getItemIDList* is invoked on. If the same browse for folder object is used on another thread than the thread it was instantiated on, the methods of the object may fail unless the COM library is initialized on the other thread. The *initCOM* method is provided to perform the initialization, and the *releaseCOM* method is provided to release the COM library on the other thread.

For each invocation of *initCOM* that returns true, a matching call to *releaseCOM* must be made on the same thread. Note that the *getFolder* and *getItemIDList* methods release the COM library on the thread they are executing in when the *reuse* arguemnt is false. Therefore, if *initCOM* is invoked on thread xx, the matching release COM requirement can be fulfilled by invoking *getFolder* with the *reuse* argument set to false.

If *initCOM* returns false, then there is no need to invoke *releaseCOM*. Indeed, invoking *releaseCOM* will be a mistake as this will cause the matching *initCOM* / *releaseCOM* calls to become unbalanced.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows a browse for folder object that is reused for the life-time of the dialog. In the onBrowse() method, if the method is not running on the same thread as the BrowseForFolder object was instantiated on, then COM is initialized and released each time the object is used. If it is the same thread, COM is not initialized and not released.

Finally, during the leaving() method, if the method is running on the same thread as the browse for folder was instantiated on, the item *ID* list and COM are released.

Note that this example does run correctly. But, if it were to turn out that the leaving() method was not running on the same thread as the define() method had run on, then COM and the item ID list would not be released correctly. It was only after some testing that it was determined that the program ran correctly:

```
::method defineDialog
  expose bff pidl

  ...

  say 'defineDialog() thread ID' .DlgUtil~threadID
  say
  bff = .BrowseForFolder~new
  bff~usePathForHint = .true
  bff~hint = ''

  pidl = .nil

::method onBrowse unguarded
  expose bff pidl

  say 'onBrowse() thread:' .DlgUtil~threadID
  say

  newThread = (bff~initialThread == .DlgUtil~threadID)

  if newThread then bff~initCOM

  if pidl \== .nil then bff~releaseItemIDList(pidl)
  if bff~owner \== .nil then bff~owner = self

  pidl = bff~getItemIDList(.true)

  if pidl \== .nil then say 'The user picked:' bff~getDisplayName(pidl)
  else say 'The user canceled'
  say

  if newThread then bff~releaseCOM

  return 0

::method leaving unguarded
  expose bff pidl

  if bff~initialThread == .DlgUtil~threadID then do
      say 'leaving() thread ID' .DlgUtil~threadID
      if pidl \== .nil then bff~releaseItemIDList(pidl)
      bff~releaseCOM
  end

/* Output from a typical session:

defineDialog() thread ID 4032

onBrowse() thread: 3320

The user picked: C:\Rexx\ooRexx.3.0.0.release

onBrowse() thread: 3320

The user canceled

onBrowse() thread: 3320

The user picked: U:\PictureFiles

leaving() thread ID 4032

*/
```

32.4.17. releaseItemIDList

```
>>--releaseItemIDList(--itemIDList--)------------><
```

Releases the system resources used by an item *ID* list.

Arguments:

The single argument is;

itemIDList [required]

The item ID list to be released.

Return value:

Returns 0, always.

Remarks:

An item ID list is similar to a *handle* to other system resources, like a *font* or a *device context*, and must be released to free up the resource. Each item ID list returned from the *getItemIDList* method should be released when the application is finished using it, unless the item ID list is assigned to the *root* attribute. **Note** that the *releaseItemIDList* does not need the COM library to be initialized. Therefore there is no problem invoking *releaseItemIDList after* the COM library has been released.

The one caveat to the rule that every item ID list must be released is, if the returned item ID list is set as the *root* attribute, there is no need to release that itme ID list, the ooDialog framework will release it automatically.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example configures a browse for folder dialog to allow browsing the entire shell namespace. It then runs in a loop allowing the user to pick a folder and see what the folder's display name is. Note that every returned item ID list is released. Also note that the *getItemIDList* method uses true for the *reuse* argument. This tells the ooDialog framework to not release COM during the invocation of the *getItemIDList* method. Because of this the example uses *releaseCOM* to properly release the COM library when the example ends:

```
bff = .BrowseForFolder~new
bff~options = '';
bff~root = 'CSIDL_DESKTOP'

msgQuestion = .endOfLine~copies(2) || "Do you want to continue browsing?"
title = 'Browsing All Shell Folders'

yes = .PlainBaseDialog~IDYES
keepGoing = yes
do while keepGoing == yes
    idList = bff~getItemIDList(.true)

    if idList == .nil then do
        nop
    end
    else do
        msg = 'You picked the "'                          || -
              bff~getDisplayName(idList, 'NORMALDISPLAY') || -
              '" folder' || msgQuestion

        keepGoing = MessageDialog(msg, , title, 'YESNO', 'QUESTION')

        bff~releaseItemIDList(idList)
    end
end

bff~releaseCom

return 0
```

```
::requires 'ooDialog.cls'
```

32.5. ComConstants Mixin Class

The **`ComConstants`** class defines **`constant`** methods useful when working with the shell objects in ooDialog. It is a *mixin* class that is inherited by the *ShellItemFilter* and *CommonDialogEvents* classes. The ooDialog programmer may also want her own classes to inherit the **`ComConstants`** class.

The current constant values represent COM result code values. These result codes may be returned by the operating system to the Rexx program or returned to the operating system from the Rexx program. The exact meaning of the result code is usually dependent on the context it is used in. This meaning will be documented in the individual shell object methods or event handlers.

32.5.1. Constant Method Table

The following table lists the constant methods of the **`ComConstants`** class:

Table 32.3. ComConstants Class Constant Method Reference

Constant	Description
CANCELED	The return value from the *show* method of the *CommonItemDialog* class when the user cancels the dialog.
E_NOTIMPL	The particular method of the COM object is not implemented.
S_FALSE	False, not true.
S_OK	The operation completed successfully, or without errors.

32.6. CommonDialogCustomizations Class

The **`CommonDialogCustomizations`** class is a *mixin* class that provides a way to customize a *CommonItemDialog* object. The **`CommonDialogCustomizations`** class has methods that allow an application to add controls to a common item dialog. To use this *mixin* class, the programmer **must** add it to the common item dialog's inheritance chain before instantiating a *OpenFileDialog* or *SaveFileDialog* object as shown in this code snippet:

```
.CommonItemDialog~inherit(.CommonDialogCustomizations)

sfd = .SaveFileDialog~new

-- Customize the dialog
ret = sfd~startVisualGroup(IDC_GB_RADIOS, 'Save file with extension:')
ret = sfd~addRadioButtonList(IDC_RBL_RADIOS)
...
```

Controls should be added to the dialog before the dialog is shown. The layout of the controls is implied by the order in which they are added. They can be added above or below the standard file dialog's set of controls. Once the dialog is shown, controls cannot be added or removed, but the existing controls can be hidden or disabled at any time. Their labels can also be changed at any time.

Container controls are controls that can have items added to them. Container controls include combo boxes, menus, drop-down lists attached to the Open or Save button, and any radio button groups. The order that items appear in a container is the order in which they were added. The operating system does not provide a way to reorder them. IDs are scoped to the parent control. Container controls, with the exception of menus, have a selected item.

Items with a container control cannot be changed after they have been created, except for their enabled and visible states. However, they can be added and removed at any time. For example, if the programmer needed to change the text of a menu, she would have to remove the current menu and add another with the correct text.

32.6.1. Method Table

The following table lists the class and instance methods of the **`CommonDialogCustomizations`** class:

Table 32.4. CommonDialogCustomizations Class Method Reference

Method	Description
Instance Methods	**Instance Methods**
addCheckButton	Adds a check box to the dialog.
addComboBox	Adds a combo box to the dialog.
addControlItem	Adds a control item to a container control in the dialog.
addEditBox	Adds an edit box control to the dialog.
addMenu	Adds a menu to the dialog.
addPushButton	Adds a push button to the dialog.
addRadioButtonList	Adds a radio button group to the dialog.
addSeparator	Adds a separator to the dialog, allowing a visual separation of controls.
addText	Adds text content, ie. a static text control, to the dialog.
check	Sets the state of the specified check box to *checked*.
enableOpenDropDown	Enables a drop-down list on the Open or Save button in the dialog.
endVisualGroup	Stops the addition of elements to a visual group in the dialog.
getCheckButtonState	Determines the state of the specified check box, checked or not checked.
getControlItemState	Gets the current state of an item in a container control found in the dialog.
getControlState	Gets the current visibility and enabled states of the specified control.
getEditBoxText	Gets the current text in an edit box control.
getSelectedControlItem	Gets the ID of the selected item in the specified specified container control in the dialog.
isChecked	Returns true if the specified check box is checked and false if it is not checked.
makeProminent	Places a control in the dialog so that it stands out compared to other added controls.
removeAllControlItems	Removes all items from a container control in the dialog.
removeControlItem	Removes an item from a container control in the dialog.
setCheckButtonState	Sets the state of a check box, checked or not checked, in the dialog.
setControlItemState	Sets the current state of an item in a container control found in the dialog.

Method	Description
setControlItemText	Sets the text, the label, of a control item. For example, the text that accompanies a radio button or an item in a menu.
setControlLabel	Sets the text associated with a control, such as button label or a static text label.
setControlState	Sets the current visibility and enabled states of a given control.
setEditBoxText	Sets the text in an edit box control found in the dialog.
setSelectedControlItem	Sets the specified item as the selected item in a radio button group or a combo box found in the dialog.
startVisualGroup	Declares a visual group in the dialog. Subsequent calls to any *add* methods adds those controls to this group.
uncheck	Sets the state of the specified check box to *unchecked*.

32.6.2. addCheckButton

```
>>--addCheckButton(--id--,--label--+------------+--)------------><
                                   +-,-checked--+
```

Adds a check box to the dialog.

Arguments:

The arguments are:

id [required]

The resource ID of the check box control. May be numeric or *symbolic*.

label [required]

The text, the label, for the check box.

checked [optional]

True if the check box should be checked, false if it should not be checked. The default if this argument is omitted is false.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The default state for the check box is enabled and visible.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example adds a checked check box to the save file dialog:

```
.CommonItemDialog~inherit(.CommonDialogCustomizations)

sfd = .SaveFileDialog~new

-- Customize the dialog
```

```
    ret = sfd~startVisualGroup(IDC_GB_FORMAT, 'Save file format:')
    ret = sfd~addCheckBox(IDC_CKL_UNICODE, 'Unicode', .true)
    ...
```

32.6.3. addComboBox

```
>>--addComboBox(--id--)--------------------------><
```

Adds a combo box to the dialog.

Arguments:

The single argument is:

id [required]

The resource ID of the combo box control. May be numeric or *symbolic*.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The default state for the check box is enabled and visible.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example adds a combo box to a save file dialog:

```
    .CommonItemDialog~inherit(.CommonDialogCustomizations)

    sfd = .SaveFileDialog~new

    -- Customize the dialog
    ret = sfd~addComboBox(IDC_CB)
    ...
```

32.6.4. addControlItem

```
>>--addControlItem(--containerID--,--label--,--itemID--)--------><
```

Adds a control item to a container control in the dialog.

Arguments:

The arguments are:

containerID [required]

The resource ID of the container to which the control item is being added. May be numeric or *symbolic*.

label [required]

The label, the text, for the added control item itemID [required]

The resource ID of the control item. May be numeric or *symbolic*.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The default state for the added item is enabled and visible. Control items in container control groups cannot be changed after they have been created, with the exception of their enabled and visible states. Container controls include radio button groups, combo boxes, drop-down lists on the Open or Save button, and menus.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example uses the *addControlItem* method to add 3 menu items to a menu:

```
.CommonItemDialog~inherit(.CommonDialogCustomizations)

sfd = .SaveFileDialog~new

-- Customize the dialog
ret = sfd~addMenu(IDC_MENU, "Pick from this Menu")
ret = sfd~addControlItem(IDC_MENU, 'Select Rexx', IDC_MENU_REX)
ret = sfd~addControlItem(IDC_MENU, 'Use text', IDC_MENU_TXT)
ret = sfd~addControlItem(IDC_MENU, 'Write to console', IDC_MENU_CONSOLE)
...
```

32.6.5. addEditBox

```
>>--addEditBox(--id--,--+---------+-)-----------><
                        +-,-text--+
```

Adds an edit box control to the dialog.

Arguments:

The arguments are:

id [required]

The resource ID of the edit control. May be numeric or *symbolic*.

text [optional]

Text to be placed in the edit box.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The default state for this control is enabled and visible. To add a label next to the edit box, place it in a visual group by using the *startVisualGroup* method.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This first example adds an empty edit box with a label, *Some text please:*, next to it:

```
.CommonItemDialog~inherit(.CommonDialogCustomizations)

sfd = .SaveFileDialog~new

-- Customize the dialog
...
ret = sfd~startVisualGroup(IDC_VG_EDIT, 'Some text please:')
ret = sfd~addEditBox(IDC_EDIT)
ret = sfd~endVisualGroup()
ret = sfd~addSeparator(IDC_SEPARATOR1)
```

This second example adds an edit box with a label, *Some text please:*, next to it. The edit box will contain the string *Useful text only*:

```
.CommonItemDialog~inherit(.CommonDialogCustomizations)

sfd = .SaveFileDialog~new

-- Customize the dialog
...
ret = sfd~startVisualGroup(IDC_VG_EDIT, 'Some text please:')
ret = sfd~addEditBox(IDC_EDIT, 'Useful text only')
ret = sfd~endVisualGroup()
ret = sfd~addSeparator(IDC_SEPARATOR1)
```

32.6.6. addMenu

```
>>--addMenu(--id--,--label--)--------------------><
```

Adds a menu to the dialog.

Arguments:

The arguments are:

id [required]

The resource ID of the menu control. May be numeric or *symbolic*.

label [required]

The label for the menu control.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The default state for the menu control is enabled and visible. Menus are container controls. To add menu items to the menu use the *addControlItem* method.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example uses the *addMenu* method to add a menu to the dialog:

```
.CommonItemDialog~inherit(.CommonDialogCustomizations)

sfd = .SaveFileDialog~new

-- Customize the dialog
ret = sfd~addMenu(IDC_MENU, "Pick from this Menu")
ret = sfd~addControlItem(IDC_MENU, 'Select Rexx', IDC_MENU_REX)
ret = sfd~addControlItem(IDC_MENU, 'Use text', IDC_MENU_TXT)
ret = sfd~addControlItem(IDC_MENU, 'Write to console', IDC_MENU_CONSOLE)
...
```

32.6.7. addPushButton

```
>>--addPushButton(--id--,--label--)--------------><
```

Adds a push button to the dialog.

Arguments:

The arguments are:

id [required]

The resource ID of the push button. May be numeric or *symbolic*.

label [required]

The label for the push button.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The default state for the push button is enabled and visible.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example adds a push button to a save file dialog:

```
.CommonItemDialog~inherit(.CommonDialogCustomizations)

sfd = .SaveFileDialog~new

-- Customize the dialog
...
sfd~addPushButton(IDC_PB_DONE, 'Done Now')
sfd~makeProminent(IDC_PB_DONE)
```

32.6.8. addRadioButtonList

```
>>--addRadioButtonList(--id--)-------------------><
```

Adds a radio button group to the dialog.

Arguments:

The single argument is:

id [required]

The resource ID of the radio button group. May be numeric or *symbolic*.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The default state for a radio button group control is enabled and visible. Radio button groups are container controls. Use the *addControlItem* method to add radio buttons to the group. To have a label for the radio button group, use the *startVisualGroup* method.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example adds a radio button group with 3 radio buttons and a label:

```
.CommonItemDialog~inherit(.CommonDialogCustomizations)

sfd = .SaveFileDialog~new

-- Customize the dialog
ret = sfd~startVisualGroup(IDC_GB_RADIOS, 'Save file with extension:')
ret = sfd~addRadioButtonList(IDC_RBL_RADIOS)
ret = sfd~addControlItem(IDC_RBL_RADIOS, '*.rex', IDC_RB_REX)
ret = sfd~addControlItem(IDC_RBL_RADIOS, '*.txt', IDC_RB_TXT)
ret = sfd~addControlItem(IDC_RBL_RADIOS, '*.rxg', IDC_RB_RXG)
ret = sfd~endVisualGroup()
...
```

32.6.9. addSeparator

```
>>--addSeparator(--id--)-------------------------><
```

Adds a separator to the dialog, allowing a visual separation of controls.

Arguments:

The single argument is:

id [required]

The resource ID of the radio button group. May be numeric or *symbolic*.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The default state for the separator control is enabled and visible.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example adds a separator control to separate the edit control from the menu below it:

```
.CommonItemDialog~inherit(.CommonDialogCustomizations)

sfd = .SaveFileDialog~new

-- Customize the dialog
...
ret = sfd~startVisualGroup(IDC_VG_EDIT, 'Some text please:')
ret = sfd~addEditBox(IDC_EDIT)
ret = sfd~endVisualGroup()
ret = sfd~addSeparator(IDC_SEPARATOR1)

ret = sfd~addMenu(IDC_MENU, "Pick from this Menu")
```

32.6.10. addText

```
>>--addText(--id--,--text--)---------------------><
```

Adds text content, ie. a static text control, to the dialog.

Arguments:

The arguments are:

id [required]

The resource ID of the text control. May be numeric or *symbolic*.

text [required]

The text to add.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The default state for the text control is enabled and visible.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example adds a static text control to the dialog right below a separator:

```
.CommonItemDialog~inherit(.CommonDialogCustomizations)

sfd = .SaveFileDialog~new

-- Customize the dialog
...
ret = sfd~addSeparator(IDC_SEPARATOR2)
ret = sfd~addText(IDC_TEXT, "No files are selected")
...
```

32.6.11. check

```
>>--check(--id--)--------------------------------><
```

Sets the state of the specified check box to *checked*.

Arguments:

The single argument is:

id [required]

The resource ID of the check box. May be numeric or *symbolic*.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

This is a convenience method. It is equivalent to using the *setCheckButtonState* method with a second arugment of *.true*.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

32.6.12. enableOpenDropDown

```
>>--enableOpenDropDown(--id--)-------------------><
```

Enables a drop-down list on the Open or Save button in the dialog.

Arguments:

The single argument is:

id [required]

The resource ID of the drop-down list. May be numeric or *symbolic*.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

This method applies to both the Open file dialog and to the Save file dialog. That is, for an Open file dialog it enables a drop-down list on the Open button and for a Save file dialog it enables a drop-down list on the Save button. After enabling the drop-down list use the *addControlItem* method to add items to the list.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example enables a drop-down list on the Save button of the Save file dialog. It then adds 3 items to the list:

```
.CommonItemDialog~inherit(.CommonDialogCustomizations)

sfd = .SaveFileDialog~new

-- Customize the dialog
...
ret = sfd~enableOpenDropDown(IDC_DROP)
ret = sfd~addControlItem(IDC_DROP, 'Always overwrite', IDC_DROP_OVERWRITE)
ret = sfd~addControlItem(IDC_DROP, 'Prompt on overwrite', IDC_DROP_PROMPT)
ret = sfd~addControlItem(IDC_DROP, 'Never overwrite', IDC_DROP_NOOVERWRITE)
```

32.6.13. endVisualGroup

```
>>--endVisualGroup-------------------------------><
```

Stops the addition of elements to a visual group in the dialog.

Arguments:

There are no arguments to this method.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The *startVisualGroup* method begins a visual group. All controls added after the invocation of that method are added to that visual group until the *endVisualGroup* method is invoked.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example starts a visual group, adds a radio button list with 3 radio buttons, and then ends the visual group:

```
.CommonItemDialog~inherit(.CommonDialogCustomizations)

sfd = .SaveFileDialog~new

-- Customize the dialog
ret = sfd~startVisualGroup(IDC_GB_RADIOS, 'Save file with extension:')
ret = sfd~addRadioButtonList(IDC_RBL_RADIOS)
ret = sfd~addControlItem(IDC_RBL_RADIOS, '*.rex', IDC_RB_REX)
```

```
    ret = sfd~addControlItem(IDC_RBL_RADIOS, '*.txt', IDC_RB_TXT)
    ret = sfd~addControlItem(IDC_RBL_RADIOS, '*.rxg', IDC_RB_RXG)
    ret = sfd~endVisualGroup()
    ...
```

32.6.14. getCheckButtonState

```
>>--getCheckButtonState(--id--)------------------><
```

Determines the state of the specified check box, checked or not checked.

Arguments:

The single argument is:

id [required]

The resource ID of the drop-down list. May be numeric or *symbolic*.

Return value:

Returns true if the check box is checked, otherwise false. If an error were to happen, the **`.SystemErrorCode`** will be set.

Remarks:

The *isChecked* method is an alias for this method.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

32.6.15. getControlItemState

```
>>--getControlItemState(--containerID--,--itemID--)-------------><
```

Gets the current state of an item in a container control found in the dialog.

Arguments:

The arguments are:

containerID [required]

The resource ID of the container the control item is in. May be numeric or *symbolic*.

itemID [required]

The resource ID of the control item. May be numeric or *symbolic*.

Return value:

Returns exactly one of the following keyword strings, or the **`.nil`** object on error. If an error were to happen, the **`.SystemErrorCode`** will be set:

Inactive:

The control is not accessible by the user. It is hidden and disabled.

Enabled:

The control is enabled, but not visible.

Visible:

The control is visible, but disabled.

EnabledVisible

The control is visible and enabled.

Remarks:

The default state of a control item is enabled and visible. Items in control groups cannot be changed after they have been created, with the exception of their enabled and visible states. Container controls include radio button groups, combo boxes, drop-down lists on the Open or Save button, and menus.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example shows a *onFileOk* event handler that first checks if the **`.rex`** radio button has not been hidden or disabled. If so, it then checks if that radio button is the selected radio button in a radio button list:

```
::class 'CDEvents' subclass CommonDialogEvents

::method onFileOk unguarded
  use arg cfd, hwnd

  state = cfd~getControlItemState(IDC_RBL_RADIOS, IDC_RB_REX)
  if state == 'EnabledVisible' & state \== .nil then do
    -- See if .rex is selected
    if cfd~getSelectedControlItem(IDC_RBL_RADIOS) == .constDir[IDC_RB_REX] then do
      -- Do something ...
    end
    else do
      -- Do some other thing
    end
  end

  return self~S_OK
```

32.6.16. getControlState

```
>>--getControlState(--id--)----------------------><
```

Gets the current visibility and enabled states of the specified control.

Arguments:

The single argument is:

id [required]

The resource ID of the control whose state is being sought. May be numeric or *symbolic*.

Return value:

Returns exactly one of the following keyword strings, or the **`.nil`** object on error. If an error were to happen, the **`.SystemErrorCode`** will be set:

Inactive:
: The control is not accessible by the user. It is hidden and disabled.

Enabled:
: The control is enabled, but not visible.

Visible:
: The control is visible, but disabled.

EnabledVisible
: The control is visible and enabled.

Remarks:

This method is very similar to the *getControlItemState* method, but it is used for controls that are *not* in a container.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

32.6.17. getEditBoxText

```
>>--getEditBoxText(--id--)-----------------------><
```

Gets the current text in an edit box control.

Arguments:

The single argument is:

id [required]
: The resource ID of the edit box. May be numeric or *symbolic*.

Return value:

Returns the text in the specified edit box, the **.nil** object on error. On error the **.SystemErrorCode** will be set.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example if from an application that saves a comment with each file when it is saved. When the save file dialog is ended, the application checks that the comment is not blank:

```
::class 'CDEvents' subclass CommonDialogEvents

::method onFileOk unguarded
  use arg cfd, hwnd

  text = cfd~getEditBoxText(IDC_EDIT); say 'text:' text
  if text == '' then do
    title = 'Save File Error'
    msg = 'You must enter a comment when saving this'.endOfLine || -
```

```
            'file. The comment will be saved along with the file.'

      j = MessageDialog(msg, hwnd, title, OK, ERROR)
      return self~S_FALSE
    end

    return self~S_OK
```

32.6.18. getSelectedControlItem

```
>>--getSelectedControlItem(--idContainer--)---------------------------------------------><
```

Gets the ID of the selected item in the specified specified container control in the dialog.

Arguments:

The arguments are:

idContainer [required]

The resource ID of the container control. May be numeric or *symbolic*.

Return value:

Returns the ID of the control item in the container that is the current selected item, or the **`.nil`** object on error.

Remarks:

To determine the user's final choice, this method can be called on radio button groups, combo boxes, and drop-down lists on the Open or Save button *after* the dialog has closed. This method cannot be called on menus. For radio button groups and combo boxes, this method *can also* be called while the dialog is showing, to determine the current choice.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example gets the selected radio button in a radio button list to test if the selection is the .rex radio button:

```
::class 'CDEvents' subclass CommonDialogEvents

::method onFileOk unguarded
  use arg cfd, hwnd

  state = cfd~getControlItemState(IDC_RBL_RADIOS, IDC_RB_REX)
  if state == 'EnabledVisible' & state \== .nil then do
    -- See if .rex is selected
    if cfd~getSelectedControlItem(IDC_RBL_RADIOS) == .constDir[IDC_RB_REX] then do
      -- Do something ...
    end
    else do
      -- Do some other thing
    end
  end

  return self~S_OK
```

32.6.19. isChecked

```
>>--isChecked(--id--)----------------------------><
```

Returns true if the specified check box is checked and false if it is not checked.

Arguments:

The single argument is:

id [required]
    The resource ID of the check box. May be numeric or *symbolic*.

Return value:

Returns true if the check box is checked, otherwise false. If an error were to happen, the `.SystemErrorCode` will be set.

Remarks:

The *getCheckButtonState* method is equivalent to this method.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

32.6.20. makeProminent

```
>>--makeProminent(--id--)------------------------><
```

Places a control in the dialog so that it stands out compared to other added controls.

Arguments:

The single argument is:

id [required]
    The resource ID of the control to be made prominent. May be numeric or *symbolic*.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

This method causes the control to be placed near the Open or Save button instead of being grouped with the rest of the custom controls. Only check boxes, push buttons, combo boxes, and menus, or a visual group that contains only a single item of one of those types, can be made prominent.

Only one control can be marked in this way. If a dialog has only one added control, that control is marked as prominent by default.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example adds a push button and then a bunch of other controls. As a last step it uses the *makeProminent* method on the push button. This places the push button next to the Save button rather than at the beginning of the controls:

```
  .CommonItemDialog~inherit(.CommonDialogCustomizations)

  sfd = .SaveFileDialog~new

  -- Customize the dialog
  sfd~addPushButton(IDC_PB_DONE, 'Done Now')
  ret = sfd~startVisualGroup(IDC_GB_RADIOS, 'Save file with extension:'); say
 'startVisualGroup:' .DlgUtil~errMsg(ret)
  ret = sfd~addRadioButtonList(IDC_RBL_RADIOS); say
 'addRadioButtonList:' .DlgUtil~errMsg(ret)
  ret = sfd~addControlItem(IDC_RBL_RADIOS, '*.rex', IDC_RB_REX); say
 'addControlItem:' .DlgUtil~errMsg(ret)
  ret = sfd~addControlItem(IDC_RBL_RADIOS, '*.txt', IDC_RB_TXT); say
 'addControlItem:' .DlgUtil~errMsg(ret)
  ret = sfd~addControlItem(IDC_RBL_RADIOS, '*.rxg', IDC_RB_RXG); say
 'addControlItem:' .DlgUtil~errMsg(ret)
  ret = sfd~endVisualGroup(); say 'endVisualGroup:' .DlgUtil~errMsg(ret)
  ...
  -- Set the initial state
  ret = sfd~setFileTypeIndex(1)
  ret = sfd~setSelectedControlItem(IDC_RBL_RADIOS, IDC_RB_REX)

  sfd~makeProminent(IDC_PB_DONE)
```

32.6.21. removeAllControlItems

```
>>--removeAllControlItems(--idContainer--)-------><
```

Removes all items from a container control in the dialog.

Arguments:

The arguments are:

idContainer [required]

The resource ID of the container control. May be numeric or *symbolic*.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

Container controls include radio button groups, combo boxes, drop-down lists on the Open or Save button, and menus. Testing has shown that the operating system returns *E_NOTIMPL* when this method is invoked on the COM object. It may be that the COM object implements the method in later versions of Windows.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

32.6.22. removeControlItem

```
>>--removeControlItem(--containerID--,--itemID--)---------------><
```

Removes an item from a container control in the dialog.

Arguments:

The arguments are:

containerID [required]

The resource ID of the container to which the control item is being removed. May be numeric or *symbolic*.

itemID [required]

The resource ID of the control item. May be numeric or *symbolic*.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

Container controls include radio button groups, combo boxes, drop-down lists on the Open or Save button, and menus. Testing has shown that the operating system returns *E_NOTIMPL* when this method is invoked on radio button list COM container object. But, it seems to work fine on other containers. It may be that the radio button list COM object implements the method in later versions of Windows.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example removes all the menu items from a menu by removing them one by one:

```
  ret = cfd~removeControlItem(IDC_MENU, IDC_MENU_REX)
  ret = cfd~removeControlItem(IDC_MENU, IDC_MENU_TXT)
  ret = cfd~removeControlItem(IDC_MENU, IDC_MENU_CONSOLE)
```

32.6.23. setCheckButtonState

```
>>--setCheckButtonState(--id--,--checked--)------><
```

Sets the state of a check box, checked or not checked, in the dialog.

Arguments:

The arguments are:

id [required]

The resource ID of the check box. May be numeric or *symbolic*.

checked [required]

True or false. True to set the check box state to checked and false to set it to unchecked.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The *check* and *uncheck* methods are convenience methods that produce similar results and eliminate the need to supply a second argument.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

32.6.24. setControlItemState

```
>>--setControlItemState(--containerID--,--itemID--,--state--)---><
```

Sets the current state of an item in a container control found in the dialog.

Arguments:

The arguments are:

containerID [required]

The resource ID of the container in which the state of the control item is being set. May be numeric or *symbolic*.

itemID [required]

The resource ID of the control item. May be numeric or *symbolic*.

state [required]

Exactly one of the following keywords that specifies the state to set, case is not significant:

INACTIVE	VISIBLE
ENABLED	ENABLEDVISIBLE

INACTIVE

Set the control item's state to disabled and hidden.

ENABLED

Set the control item's state to enabled. It will not be visible to the user.

VISIBLE

Set the control item's state to visible, but it will not be enabled.

ENABLEDVISIBLE

Set the control item's state to enabled and visible.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The default state of a control item is enabled and visible. Items in control groups cannot be changed after they have been created, with the exception of their enabled and visible states. Container controls include radio button groups, combo boxes, drop-down lists on the Open or Save button, and menus.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example hides and disables one of the menu items in a menu:

```
ret = cfd~setControlItemState(IDC_MENU, IDC_MENU_REX, 'INACTIVE')
```

32.6.25. setControlItemText

```
>>--setControlItemText(--containerID--,--itemID--,--text--)-----><
```

Sets the text, the label, of a control item. For example, the text that accompanies a radio button or an item in a menu.

Arguments:

The arguments are:

containerID [required]
: The resource ID of the container in which the text of the control item is being set. May be numeric or *symbolic*.

itemID [required]
: The resource ID of the control item. May be numeric or *symbolic*.

text [required]
: The text to set the label of the control item to.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example changes the labels of 3 menu items in a menu:

```
ret = cfd~setControlItemText(IDC_MENU, IDC_MENU_REX, 'Pick Rexx')
ret = cfd~setControlItemText(IDC_MENU, IDC_MENU_TXT, 'Pick Text')
ret = cfd~setControlItemText(IDC_MENU, IDC_MENU_CONSOLE, 'Pick The Console')
```

32.6.26. setControlLabel

```
>>--setControlLabel(--id--,--text--)-------------><
```

Sets the text associated with a control, such as button label or a static text label.

Arguments:

The arguments are:

id [required]

The resource ID of the control May be numeric or *symbolic*.

text [required]

The text to associate with the control.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example changes the labels of a push button and a static text control to reflect a change in the function of saving a file:

```
ret = cfd~setControlLabel(IDC_PB_DONE, 'Save / Backup')
ret = cfd~setControlLabel(IDC_TEXT, 'When Saving a file a backup file will be created')
```

32.6.27. setControlState

```
>>--setControlState(--id--,--state--)------------><
```

Sets the current visibility and enabled states of a given control.

Arguments:

The arguments are:

id [required]

The resource ID of the control whose state is being set. May be numeric or *symbolic*.

state [required]

Exactly one of the following keywords that specifies the state to set, case is not significant:

INACTIVE	VISIBLE
ENABLED	ENABLEDVISIBLE

INACTIVE

Set the control's state to disabled and hidden.

ENABLED

Set the control's state to enabled. It will not be visible to the user.

VISIBLE

Set the control's state to visible, but it will not be enabled.

ENABLEDVISIBLE
: Set the control's state to enabled and visible.

Return value:
: Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:
: When the dialog is shown, controls cannot be added or removed, but the existing controls can be hidden or disabled at any time.

Details
: Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:
: This example disables an extra push button that was added to the dialog when the function of the push button is not available:

```
ret = cfd~setControlState(IDC_PB_DONE, 'VISIBLE')
```

32.6.28. setEditBoxText

```
>>--setEditBoxText(--id--,--text--)--------------><
```

Sets the text in an edit box control found in the dialog.

Arguments:
: The arguments are:

id [required]
: The resource ID of the edit box. May be numeric or *symbolic*.

text [required]
: The text to put in the edit box.

Return value:
: Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Details
: Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:
: This example comes from an application that creates a back up file each time the user saves a file. The Save File dialog automatically determines a back up file name, but the user is allowed to change the name. The application puts the suggested back up name in an edit box, which the user can edit if desired:

```
buName = self~calcBackupName()
cfd~setEditBoxText(IDC_EDIT_BU_NAME, buName)
```

32.6.29. setSelectedControlItem

```
>>--setSelectedControlItem(--containerID--,--itemID--)----------><
```

Sets the specified item as the selected item in a radio button group or a combo box found in the dialog.

Arguments:

The arguments are:

containerID [required]

The resource ID of the container whose selected item is being set. May be numeric or *symbolic*.

itemID [required]

The resource ID of the control item that will be set to the selected item. May be numeric or *symbolic*.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

Although it seems as though this method should work with drop-down lists on the Open or Save button, testing has shown that it does not. The underlying COM object returns *error code 0x80004001, Not implemented*

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example adds a radio button group to a Save File dialog and then sets the middle radion button as the selected item:

```
.CommonItemDialog~inherit(.CommonDialogCustomizations)

sfd = .SaveFileDialog~new

-- Customize the dialog
ret = sfd~startVisualGroup(IDC_GB_RADIOS, 'Save file with extension:')
ret = sfd~addRadioButtonList(IDC_RBL_RADIOS)
ret = sfd~addControlItem(IDC_RBL_RADIOS, '*.txt', IDC_RB_TXT)
ret = sfd~addControlItem(IDC_RBL_RADIOS, '*.rex', IDC_RB_REX)
ret = sfd~addControlItem(IDC_RBL_RADIOS, '*.rxg', IDC_RB_RXG)
ret = sfd~endVisualGroup()

ret = sfd~setSelectedControlItem(IDC_RBL_RADIOS, IDC_RB_REX)
```

32.6.30. startVisualGroup

```
>>--startVisualGroup(--id--,--label--)------------------------------------------><
```

Declares a visual group in the dialog. Subsequent calls to any *add* methods adds those controls to this group.

Arguments:

The arguments are:

id [required]

The resource ID of the visual group. May be numeric or *symbolic*.

label [required]

A lable that is used with the visual group.

Return value:

Additional comments.

Remarks:

Controls will continue to be added to this visual group until the *endVisualGroup* method is invoked.

A visual group can be hidden and disabled like any other control, except that doing so affects all of the controls within it. The control members of the visual group can also be hidden and disabled individually.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example starts a visual group that consists of a radio button group. One main reason for starting a visual group, rather than just adding the radion button group, is that the visual group makes it easy to add a label for the radio button group:

```
.CommonItemDialog~inherit(.CommonDialogCustomizations)

sfd = .SaveFileDialog~new

-- Customize the dialog
ret = sfd~startVisualGroup(IDC_GB_RADIOS, 'Save file with extension:')
ret = sfd~addRadioButtonList(IDC_RBL_RADIOS)
ret = sfd~addControlItem(IDC_RBL_RADIOS, '*.txt', IDC_RB_TXT)
ret = sfd~addControlItem(IDC_RBL_RADIOS, '*.rex', IDC_RB_REX)
ret = sfd~addControlItem(IDC_RBL_RADIOS, '*.rxg', IDC_RB_RXG)
ret = sfd~endVisualGroup()

ret = sfd~setSelectedControlItem(IDC_RBL_RADIOS, IDC_RB_REX)
```

32.6.31. uncheck

```
>>--uncheck(--id--)------------------------------><
```

Sets the state of the specified check box to *unchecked*.

Arguments:

The single argument is:

id [required]
The resource ID of the check box. May be numeric or *symbolic*.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

This is a convenience method. It is equivalent to using the *setCheckButtonState* method with a second arugment of *.false*.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

32.7. CommonItemDialog Class

The common item dialog was formerly known as the common file dialog. It is used in two variations, the open file dialog and the save file dialog. The common item dialog is only available on a Vista or later version of Windows. The *CommonItemDialog* class provides most of the implementation of the common item dialog, but is not instantiated directly by the Rexx programmer. Rather the programmer instantiates one of its concrete subclasses, the *OpenFileDialog* class, or *SaveFileDialog* class.

The common item dialog is more full featured than the earlier common file dialog. It is also more customizable. ooDialog supplies some helper classes for the common item dialog:

- The *CommonDialogCustomizations* class: Provides a way to customize the common item dialog that is shown to the user.
- The *CommonDialogEvents* class: Provides a way to connect event notifications sent from the common item dialog.
- The *ShellItemFilter* class: Adds the ability to do special filtering of the folders and files shown to the user in the common item dialog.

32.7.1. Method Table

The following table lists the class and instance methods of the **`CommonItemDialog`** class:

Table 32.5. CommonItemDialog Class Method Reference

Method	Description
Class Methods	**Class Methods**
new	The common item dialog can not be instantiated directly. Instead instantiate an *OpenFileDialog* or a *SaveFileDialog* object.
Attribute Methods	**Attribute Methods**
options	Reflects the options in effect for the common item dialog.
Instance Methods	**Instance Methods**
addPlace	Adds a folder to the list of places available for the user to open or save items.
advise	Assigns an event handler object that invokes methods in the ooDialog program for event notificationss coming from the common item dialog.

Method	Description
clearClientData	Instructs the file dialog to clear all persistent state information.
close	Closes the dialog.
getCurrentSelection	Gets the user's current selection in the dialog.
getFileName	Retrieves the text currently entered in the dialog's file name edit box.
getFileTypeIndex	Gets the one-based index of the currently selected file type.
getFolder	Retrieves either the folder currently selected in the dialog, or, if the dialog is not currently displayed, the folder that is to be selected when the dialog is opened.
getResult	Gets the choice that the user made in the dialog.
initCOM	Initializes the COM *library* on the current thread.
isReleased	Tests if the file dialog object has been *released*.
release	Releases the system resources used by the underlying COM object that this Rexx object represents.
releaseCOM	Releases the COM library on the current thread.
setCancelButtonLabel	Sets the label of the Cancel button in the dialog.
setClientGuid	Allows the calling application to associate a GUID with a dialog's persistent state.
setDefaultExtension	Sets the default extension to be added to file names.
setDefaultFolder	Sets the folder used as a default if there is not a recently used folder value available.
setFileName	Sets the file name initially placed in the file name edit box of the dialog
setFileNameLabel	Sets the text of the label next to the file name edit box
setFileTypeIndex	Sets the file type that appears as selected in the dialog.
setFileTypes	Sets the file types filter.
setFilter	Sets a special filter that can be used to remove some items from the dialog's view.
setFolder	Sets the folder that is selected when the dialog is opened.
setOkButtonLabel	Sets the label of the Ok button in the dialog.
setTitle	Sets the title of the common item dialog.
show	Launches the modal dialog window.
unadvise	Removes the event handler that was attached through the *advise* method.

32.7.2. new (Class Method)

```
>>--new-----------------------------------------><
```

The Rexx programmer can not instantiate a **`CommonItemDialog`** directly. Rather the programmer must instantiate an *OpenFileDialog* or a *SaveFileDialog* object, depending on what functionality is desired.

32.7.3. options (Attribute)

```
>>--options-----------------------------------------------------><

>>--options = varName-------------------------------------------><
```

Reflects the options in effect for the common item dialog.

options get:

Gets the current options for the dialog as a string of space separated keyword values. The possible keywords are listed in the remarks.

options set:

Sets the current options for the dialog through the use of a string of space separated keywords, case is not significant. The current options are replaced completely by the options specified.

Remarks:

The common item dialog options control the dialog's behavior. The following lists the allowable keywords. The value of the *options* attribute is a string containing zero or more of the keywords, separated by spaces, case insignificant. As noted, some options are the default behavior of the dialog and other options can not be used together:

ALLNONSTORAGEITEMS	FORCEPREVIEWPANEON	NOTESTFILECREATE
ALLOWMULTISELECT	FORCESHOWHIDDEN	NOVALIDATE
CREATEPROMPT	HIDEMRUPLACES	OVERWRITEPROMPT
DEFAULTNOMINIMODE	HIDEPINNEDPLACES	PATHMUSTEXIST
DONTADDTORECENT	NOCHANGEDIR	PICKFOLDERS
FILEMUSTEXIST	NODEREFERENCELINKS	SHAREAWARE
FORCEFILESYSTEM	NOREADONLYRETURN	STRICTFILETYPES

ALLNONSTORAGEITEMS

Enables the user to choose any item in the shell namespace, not just those with STREAM or FILESYSTEM attributes. This flag cannot be combined with FORCEFILESYSTEM.

ALLOWMULTISELECT

Allow the user to select multiple items in the open file dialog. Note that setting this option will only have an effect on the *OpenFileDialog*.

CREATEPROMPT

Prompt for creation if the item returned in the save dialog does not exist. Note that this does not actually create the item.

DEFAULTNOMINIMODE

Indicates to the Save As dialog that it should open in expanded mode. Expanded mode is the mode that is set and unset by clicking the button in the lower-left corner of the Save As dialog that switches between *Browse Folders* and *Hide Folders* when clicked.

DONTADDTORECENT

Do not save the item being opened or saved to the list of recent places.

FILEMUSTEXIST

The item returned must exist. This is a default value for the open dialog.

FORCEFILESYSTEM
: Ensures that returned items are file system items (FILESYSTEM). Note that this does not apply to items returned by *getCurrentSelection*.

FORCEPREVIEWPANEON
: Indicates to the Open File dialog that the preview pane should always be displayed.

FORCESHOWHIDDEN
: Show hidden items.

HIDEMRUPLACES
: Hide the list of places from which the user has recently opened or saved items.

HIDEPINNEDPLACES
: Hide the list of pinned places from which the user can choose.

NOCHANGEDIR
: Not used. Note however that the operating system may return this value when querying the current options of the dialog.

NODEREFERENCELINKS
: Shortcuts should not be treated as their target items, allowing an application to open a .lnk file.

NOREADONLYRETURN
: Do not return read-only items.

NOTESTFILECREATE
: Do not test creation of the item returned in the save dialog. If this flag is not set, the calling application must handle errors discovered in the creation test.

NOVALIDATE
: Do not check for situations that would prevent an application from opening the selected file, such as sharing violations or access denied errors.

OVERWRITEPROMPT
: When saving a file, prompt before overwriting an existing file of the same name. This is a default value for the save dialog.

PATHMUSTEXIST
: The item returned must be in an existing folder. This is a default value.

PICKFOLDERS
: Enables the user to choose folders rather than files.

SHAREAWARE
: In the case of a sharing violation when an application is opening a file, call the application back through *onShareViolation* event handler for guidance. This keyword is overridden by NOVALIDATE.

STRICTFILETYPES
: Ensures that the returned file name has a file extension that matches the currently selected file type. The dialog appends the correct file extension if necessary.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows the default options in effect for a **`OpenFileDialog`**;

```
  fod = .OpenFileDialog~new

  say 'options:' fod~options

/* Output will be:
options: FILEMUSTEXIST NOCHANGEDIR PATHMUSTEXIST
*/
```

This example shows how to add additional options to the current options in effect:

```
  .CommonItemDialog~inherit(.CommonDialogCustomizations)
  fod = .OpenFileDialog~new

  fod~options = fod~options 'ALLOWMULTISELECT'
  say 'options:' fod~options

/* Output will be:
options: ALLOWMULTISELECT FILEMUSTEXIST NOCHANGEDIR PATHMUSTEXIST
*/
```

32.7.4. addPlace

```
>>--addPlace(--folder--+----------+--)-----------><
                       +-.-where--+
```

Adds a folder to the list of places available for the user to open or save items.

Arguments:

The arguments are:

folder [required]

The folder to add. The folder can be specified as the full path name of the folder, a *CSIDL_xx* name, or an *item ID list*.

where [optional]

Specifies where the folder is placed within the list. The only allowable values are TOP or BOTTOM. If the argument is omitted, the default is TOP.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

Currently the only way to get an item ID list is through the *getItemIDList* method of the *BrowseForFolder* class. Future enhancements of ooDialog may make more use of item ID list, which is why the *addPlace* method accepts an item ID list for the *folder* argument.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example adds the Fonts folder to the dialog:

```
  fod = .OpenFileDialog~new

  fod~options = 'NOCHANGEDIR PATHMUSTEXIST ALLNONSTORAGEITEMS'

  ret = fod~addPlace('CSIDL_FONTS')
```

32.7.5. advise

```
>>--advise(--eventHandler--)---------------------><
```

Assigns an event handler object that invokes methods in the ooDialog program for event notificationss coming from the common item dialog.

Arguments:

The single argument is:

eventHandler [required]

An instantiated *CommonDialogEvents* object that intercepts the common item dialog's event notifications.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

To receive event notifications from a common item dialog, the programmer creates a subclass of the **`CommonDialogEvents`** class and over-rides the event handler methods of the class for the events the programmer is interested in.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example shows a technique that allows the label of the *Cancel* button to be changed. Before the common file dialog is opened a folder changing notification is sent. The example uses an event handler method for that event and the first time it receives the notification sets the label of the cancel button to *Abort*.

```
  fileOpen = .OpenFileDialog~new

  events = .CDEvents~new
  fileOpen~advise(events)

  fileOpen~setOkButtonLabel('Open This File')

  if fileOpen~show() == .CommonItemDialog~canceled then
      say 'The user canceled'
  else
      say 'The user picked:' fileOpen~getResult

::requires 'ooDialog.cls'

::class 'CDEvents' subclass CommonDialogEvents

::method onFolderChanging unguarded
```

```
expose labelChanged
use arg cfd, hwnd

-- Only change the label on the first event.
if \ labelChanged~dataType('O') then
    labelChanged = .false

if \ labelChanged then do
    cfd~setCancelButtonLabel('Abort')
    labelChanged = .true
end

return self~S_OK
```

32.7.6. clearClientData

```
>>--clearClientData------------------------------><
```

Instructs the file dialog to clear all persistent state information.

Arguments:

There are no arguments to this method.

Return value:

Returns the *HRESULT code* that the underlying COM object returns. This method seems to always return: 0x80004005, *Unspecified error*. This is true even in the MSDN samples. However testing shows that the method works, despite the error return code.

Remarks:

In Windows XP, a state, such as the last visited folder, was saved on an application, on a per-process, basis. However, that state information was used without regard to the current action. Take an application that opens back up files and also opens text documents. If the last visited folder was the back up files folder when the user is opening up a text document, the file dialog would start in the back up files folder.

With the common item file dialog, persistent information can still be associated with an application, but it can also be associated with a GUID by using the *setClientGuid* method. If a GUID was set using the *setClientGuid* then that GUID is used to clear persisted information.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

32.7.7. close

```
>>--close(--+-----------+--)---------------------><
            +--hResult--+
```

Closes the dialog.

Arguments:

The single argument is:

hRealult [optional]

The *HRESULT code* that will be returned by the *show* method to indicate that the dialog was closed before a selection was made. This may be specified as a decimal number or as a string in conventional *hexadecimal* format. If the argument is omitted, **`S_OK`** is used.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

A programmer can invoke this method from an event handler method while the dialog is open. The dialog will close and the *show* method will return with the HRESULT specified by the *hResult* argument. If this method is invoked, there is no result available for the *getResult* or *getResults* methods, and they will fail if invoked.

Note that testing has shown that the dialog does not immediately close when the method is invoked. Rather it closes when the user selects a new file or a new folder.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example comes from a program used during the initial testing for the common item file dialog. As such it is somewhat contrived, but serves the purpose of an example. A *Closer* class is implemented with a *close* method. In the example, the *close* method is started executing on another thread. This method then sleeps for a short bit, invokes some methods on the open file dialog, and then invokes the *close* method of the open file dialog:

```
  closer = .Closer~new

  ofd = .OpenFileDialog~new

  filter = .array~of('Rexx', '*.rex', 'Bitmap', '*.bmp', 'Image', '*.jpg;*.jpeg')
  ret = ofd~setFileTypes(filter)
  ret = ofd~setFileTypeIndex(1)

  ofd~setFolder('C:\Program Files')
  closer~set(ofd)

  closer~start('close')

  ret = ofd~show()
  say 'show() ret: ' ret 'ret -> errMsg:' .DlgUtil~errMsg(ret)
  say 'getFileName:' ofd~getFileName .DlgUtil~errMsg(.systemErrorCode)
  say 'getResult:  ' ofd~getResult .DlgUtil~errMsg(.systemErrorCode)
  ofd~release

::requires 'ooDialog.cls'

::class 'Closer'

::method set
expose ofd
use strict arg ofd

::method close unguarded
expose ofd
use strict arg timeOut = 5

ofd~initCom
```

```
j = SysSleep(timeOut)
say 'current file name:      ' ofd~getFileName .DlgUtil~errMsg(.systemErrorCode)
say 'current file type index:' ofd~getFileTypeIndex .DlgUtil~errMsg(.systemErrorCode)
say 'current file selection: ' ofd~getCurrentSelection .DlgUtil~errMsg(.systemErrorCode)
say 'current folder          ' ofd~getFolder .DlgUtil~errMsg(.systemErrorCode)
say; say
ret = ofd~close('0x80004004')

ofd~releaseCom

/* Output will be something like the following:

current file name:        Error code 0 (0x00000000): The operation completed
 successfully.
current file type index: 1 Error code 0 (0x00000000): The operation completed
 successfully.
current file selection:  The NIL object Error code 2147500037 (0x80004005): Unspecified
 error
current folder           C:\Program Files Error code 0 (0x00000000): The operation
 completed successfully.


show() ret:  2147500036 ret -> errMsg: Error code 2147500036 (0x80004004): Operation
 aborted
getFileName: The NIL object Error code 2147500037 (0x80004005): Unspecified error
getResult:   The NIL object Error code 2147549183 (0x8000ffff): Catastrophic failure

*/
```

32.7.8. getCurrentSelection

```
>>--getCurrentSelection(--+--------+--)--------------------------------------------><
                          +--type--+
```

Gets the user's current selection in the dialog.

Arguments:

The single argument is:

type [optional]

A *SIGDN* keyword that specfies the format of the returned current selection name. By default if this arugment is omitted, the method uses the FILESYSPATH keyword. When used, the *type* argument specifies the format of the returned current selection name using exactly one of the *SIGDN* keywords.

Return value:

The display name of the current selection, or the **.nil** object on error.

Remarks:

This method returns the string name of the shell item currently selected in the dialog. This item can be a file or folder selected in the view window, or something that the user has typed into the dialog's edit box. The latter case may require a parsing operation (by the shell, cancelable by the user) that blocks the current thread.

Typically the *getCurrentSelection* method would be invoked from an event *handler*. Invoking the method before the file dialog is shown, or after it has been closed will return the **.nil** object.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example comes from a business application for a company that does not allow its employees to open files marked as reserved. The files are marked by the IT department by putting *resv* in the file name:

```
::method onFileOk unguarded
    use arg ofd, hwnd

    fileName = ofd~getCurrentSelection
    if fileName~caselessPos('resv') <> 0 then do
        msg = '"Reserved" files can not be opened.  Please select another file.'
        title = 'File Open Error'
        r = MessageDialog(msg, hwnd, title, 'OK', 'WARNING')
        return self~S_FALSE
    end

    return self~S_OK
```

32.7.9. getFileName

```
>>--getFileName----------------------------------><
```

Retrieves the text currently entered in the dialog's File name edit box.

Arguments:

This method does not have any arguments.

Return value:

The text in the edit box, or the **.nil** object on error.

Remarks:

Unlike some other methods such as the *getCurrentSelection*, the *getFileName* can be used after the file dialog has been closed. Even if the user canceled the dialog. Of course it can also be invoked from an event *handler*.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This code snippet shows the difference when using the *getFileName* and *getCurrentSelection* methods. Note in the output that *getFileName* returned the text in the edit box which does not match what the user actually selected as shown by *getResult*. The user selected a file by double clicking on it, which bypasses the edit box altogether:

```
  ret = ofd~show()

  say 'getFileName:        ' fod~getFileName
  say '  rc:' .DlgUtil~errMsg(.systemErrorCode)
  say
  say 'getCurrentSelection:' fod~getCurrentSelection
  say '  rc:' .DlgUtil~errMsg(.systemErrorCode)
  say
  say 'getResult:          ' fod~getResult(PARENTRELATIVE)
  say '  rc:' .DlgUtil~errMsg(.systemErrorCode)
  say
  say 'getResult:          ' fod~getResult
  say '  rc:' .DlgUtil~errMsg(.systemErrorCode)

/* output might be:

getFileName:         madeUp.name
  rc: Error code 0 (0x00000000): The operation completed successfully.

getCurrentSelection: The NIL object
  rc: Error code 2147500037 (0x80004005): Unspecified error

getResult:           Jellyfish.jpg
  rc: Error code 0 (0x00000000): The operation completed successfully.

getResult:           C:\Users\Public\Pictures\Sample Pictures\Jellyfish.jpg
  rc: Error code 0 (0x00000000): The operation completed successfully.

*/
```

32.7.10. getFileTypeIndex

```
>>--getFileTypeIndex-----------------------------><
```

Gets the one-based index of the currently selected file type.

Arguments:

This method has no arguments.

Return value:

Returns the one-based index of the selected file type in the file type array passed to the *setFileTypes* method. Returns -1 on error.

Remarks:

This method can be inovked either while the dialog is open or after it has been closed.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

32.7.11. getFolder

```
>>--getFolder(--+---------+--)-------------------><
                +--sigdn--+
```

Retrieves either the folder currently selected in the dialog, or, if the dialog is not currently displayed, the folder that is to be selected when the dialog is opened.

Arguments:

The single argument is:

sigdn [optional]
: A *SIGDN* keyword that specfies the format of the name of the currently selected folder. The default if this arugment is omitted is the FILESYSPATH keyword. If not omitted, the *sigdn* argument specifies the format of the returned folder name using exactly one of the *SIGDN* keywords.

Return value:

The currently selected folder name, or the **.nil** object on error.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example code gets the file name of the item chosen by the user and the folder the item is in:

```
  ret = fod~show()
  if ret == fod~S_OK then do
      say 'getResult:          ' fod~getResult(PARENTRELATIVE)
      say '  rc:' .DlgUtil~errMsg(.systemErrorCode)
      say

      say 'getFolder:          ' fod~getFolder
      say '  rc:' .DlgUtil~errMsg(.systemErrorCode)
      say
  end
  fod~release
  return 0

/* Output might be:
getResult:           ooRexxTry.rex
  rc: Error code 0 (0x00000000): The operation completed successfully.

getFolder:           C:\Rexx\ooRexx
  rc: Error code 0 (0x00000000): The operation completed successfully.
*/
```

32.7.12. getResult

```
>>--getResult(--+---------+--)-------------------><
                +--sigdn--+
```

Gets the choice that the user made in the dialog.

Arguments:

The single argument is:

sigdn [optional]
: A *SIGDN* keyword that specfies the format of the name of the shell item chosen by the user. The default if this arugment is omitted is the FILESYSPATH keyword. If not omitted, the *sigdn* argument specifies the format of the returned name using exactly one of the *SIGDN* keywords.

Return value:

The name of the shell item chosen by the user, or the **.nil** object on error.

Remarks:

The *getResult* method can be invoked after the dialog has closed or during the handling of an *onFileOk* event notification. Invoking this method at any other time will fail. If multiple items were chosen, this method will fail. In the case of multiple items, invoke the *getResults* method.

The *show* method must return a success code for a result to be available to the *getResult* method.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example shows how to use the *sigdn* argument to return just the file name of the item chosen by the user rather than the complete file path name:

```
  ret = fod~show()
  if ret == fod~S_OK then do
      say 'getResult:          ' fod~getResult(PARENTRELATIVE)
      say '  rc:' .DlgUtil~errMsg(.systemErrorCode)
      say

      say 'getFolder:          ' fod~getFolder
      say '  rc:' .DlgUtil~errMsg(.systemErrorCode)
      say
  end
  fod~release
  return 0

/* Output might be:
getResult:           ooRexxTry.rex
  rc: Error code 0 (0x00000000): The operation completed successfully.

getFolder:           C:\Rexx\ooRexx
  rc: Error code 0 (0x00000000): The operation completed successfully.
*/
```

32.7.13. initCOM

```
>>--initCOM-------------------------------------><
```

Initializes the COM *library* on the current thread.

Arguments:

There are no arguments for this method.

Return value:

Returns true when the COM library is initialized as a result of the *initCom* invocation. Returns false if the library is not initialized as a result of the *initCom* invocation.

Remarks:

When the common item file dialog is first instantiated, the ooDialog framework initializes the COM library on the thread that is executing. If that same object is used on another thread than the initial thread, the methods of the object may fail unless the COM library is initialized on the other thread. The *initCOM* method is provided to perform that initialization.

For each invocation of *initCOM* that returns true, a matching call to *releaseCOM* must be made on the same thread. Note that the *release* method also releases the COM library, but only *if* it is inovked on the same thread as the file dialog object was instantiated on.

If *initCOM* returns false, then there is no need to invoke *releaseCOM*. Indeed, invoking *releaseCOM* will be a mistake as this will cause the matching *initCOM* / *releaseCOM* calls to become unbalanced. When *initCom* returns false, it can be for two reasons. If the COM library has already been initialized on the current thread, false is returned and the **`.SystemErrorCode`** is set to 1. There is also the possibility of a failure in initializing the COM library. In this case, false is returned and the **`.SystemErrorCode`** is set to a COM error code returned by the COM library.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This is a contrived example that gives the user only 15 seconds to pick a file. If the user does not pick a file within that time period, the open file dialog is closed. The open file dialog object is passed to a second thread, making it necessary to initialize, and release, the COM library on that thread

```
  closer = .Closer~new

  ofd = .OpenFileDialog~new

  filter = .array~of('Rexx', '*.rex', 'Bitmap', '*.bmp', 'Image', '*.jpg;*.jpeg')
  ret = ofd~setFileTypes(filter)
  ret = ofd~setFileTypeIndex(1)

  ofd~setFolder('C:\Rexx\ooRexx')
  closer~set(ofd)

  timeout = 15
  msgObj = closer~start('close', timeout)

  ret = ofd~show()
  if ret == ofd~S_OK then do
    say 'Congrats, you picked a file within' timeout 'seconds. File:'
 ofd~getResult(PARENTRELATIVE)
  end
  else do
    say 'Tough luck, you are too slow.'
  end

  -- If we release the open file object before the
  -- second thread is finished we will have an error
  -- raised.  So we wait ...
  say 'Please wait while system resources are cleaned up ...'
  msgObj~result
  ofd~release

::requires 'ooDialog.cls'

::class 'Closer'

::method set
expose ofd
use strict arg ofd

::method close unguarded
expose ofd
use strict arg timeOut = 5
```

```
ofd~initCom

j = SysSleep(timeOut)
ret = ofd~close('0x80004004')

ofd~releaseCom
```

32.7.14. isReleased

```
>>--isReleased-----------------------------------><
```

Tests if the file dialog object has been *released*.

Arguments:

There are no arguments to this method.

Return value:

Returns true if the system resources used by the common item file dialog object have been released, otherwise false.

Remarks:

Each common item dialog that is instantiated must have its system resources released when the application is done using that object. Once the system resources for the object have been released, it is an error to invoke any method on that object and a condition will be raised if a method is invoked. The only exceptions to that rule are the *isReleased* and the *releaseCOM* methods. The *isReleased* method can be invoked at any time to test if the object has already been released.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

32.7.15. release

```
>>--release--------------------------------------><
```

Releases the system resources used by the underlying COM object that this Rexx object represents. Each instantiated common item dialog Rexx object needs to have the *release* method invoked on the object when the application is finished with the object.

Arguments:

There are no arguments for this method.

Return value:

Returns true.

Remarks:

The *release* method must be invoked once, and only once, on this object to release the system resources used by the common file dialog. This also releases the COM library, ***if*** the library was initialized on the current thread.

After *release* has been invoked, no other methods, except *isReleased* and *releaseCOM*, can be invoked on this object. The *release* method can be invoked on any thread, but this method only releases the COM library if *release* is invoked on the same thread as this object was instantiated on.

Note

It is not recommended that the programmer use a single Rexx common item dialog object on more than one thread. It is recommended that the programmer instantiate the common item dialog, configure the dialog, show the dialog, retrieve the user's input, and release the dialog, all on the same thread. It is possible to use the same object on multiple threads. However, ensuring that system resources and the COM library are cleaned up properly becomes more difficult to manage.

If the programmer uses the same common item dialog object on more than one thread, then it becomes the programmer's responsibility to ensure that the COM library is initialized and released on each thread, that the init and release are in balanced pairs, and that release is called once and only once for the object. This is certainly doable, however it is more complicated then restricting the use of the object to one thread.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example uses a single common item dialog object on two threads. It ensures that the COM library is initialized and released on each thread and that the object is released.

Note however that the code doesn't guarantee that the *getResult* or *show* is not invoked after the COM object has been released. It just makes it unlikely. If the user selects a file and the timeout happens at the same instant, both threads could receive true from *isReleased* method, because neither thread has yet invoked *release*.

```
closer = .Closer~new

ofd = .OpenFileDialog~new

filter = .array~of('Rexx', '*.rex', 'Bitmap', '*.bmp', 'Image', '*.jpg;*.jpeg')
ret = ofd~setFileTypes(filter)
ret = ofd~setFileTypeIndex(1)

ofd~setFolder('C:\Rexx\ooRexx')
closer~set(ofd)

timeout = 15
closer~start('close', timeout)

selectedFile = .nil
ret = ofd~show()
if ret == ofd~S_OK then do
  if \ ofd~isReleased then do
    selectedFile = ofd~getResult(PARENTRELATIVE)
    ofd~release
    say 'Congrats, you picked a file within' timeout 'seconds. File:' selectedFile
  end
```

```
    else do
      say 'Congrats, you picked a file within' timeout 'seconds.'
      say 'But the dialog was released before the results could be retrieved'

      -- The COM library has not been released on this thread
      ofd~releaseCOM
    end
  end
  else do
    say 'Tough luck, you are too slow.'
  end


::requires 'ooDialog.cls'

::class 'Closer'

::method set
expose ofd
use strict arg ofd

::method close unguarded
expose ofd
use strict arg timeOut = 5

ofd~initCom

j = SysSleep(timeOut)
if \ ofd~isReleased then do
  ret = ofd~close('0x80004004')
  -- Releases the object, but does not release
  -- COM on this thread, or the original thread.
  ofd~release
end

ofd~releaseCom
```

32.7.16. releaseCOM

```
>>--releaseCOM-----------------------------------><
```

Releases the COM library on the current thread.

Arguments:

There are no arguments for this method.

Return value:

Returns true if the Windows API used to release the COM library is called. Returns false if the API is not called.

Remarks:

If *releaseCom* is invoked on the same thread as this object was instantiated on, and the COM library has already been released on the thread, then the Windows API is not called and false is returned.

When the *new* method is invoked, ooDialog initializes the COM library automatically. When the *release* method is invoked, *if it is the same thread* as *new* was invoked on, ooDialog automatically releases the COM library. As long as the programmer follows the recommended practice of only using a common item dialog object on one thread, the programmer need not worry about initializing and releasing the COM library. The *releaseCOM* and *initCOM* methods are provided for the programmer's use if the recommended practice is not followed, the programmer needs to be aware of these points:

- The COM library needs to be initialized and released, once, on each thread, a single common item dialog is used on.
- If, on a thread, the library was initialized and released, and the object was going to be used again, the init / release pair would need to be done again.
- The automatic COM initialization and release done by the ooDialog framework.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

The *release() example* shows the usage of the *releaseCOM* method.

32.7.17. setCancelButtonLabel

```
>>--setCancelButtonLabel(--label--)--------------><
```

Sets the label of the Cancel button in the dialog.

Arguments:

The single argument is:

label [required]

The new label for the Cancel button.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

Unlike the *setOkButtonLabel* method, which is supplied by the underlying COM dialog object, the *setCancelButtonLabel* is an enhancement added by ooDialog. Because of this, its use is slightly different than the use of the *setOkButtonLabel* method.

The ooDialog framework gets a handle to the Cancel button and resets its label to that specified. The button does not exist until the dialog is created through the *show* method. Since the *show* method does not return until the dialog is destroyed, the programmer needs to invoke this method from an event handler in the *CommonDialogEvents* class. It has been observed that the folder *changing* event is always invoked the first time the dialog is shown, before it is visible. This makes that event handler an ideal place to change the label of the Cancel button. This technique is shown in the example. Clever programmers may implement alternative ways to use the *setCancelButtonLabel*.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example changes the labels of both the Ok and the Cancel buttons. The Cancel button is changed to *Abort*.

```
  fileOpen = .OpenFileDialog~new

  events = .CDEvents~new
  fileOpen~advise(events)

  fileOpen~setOkButtonLabel('Open This File')

  ret = fileOpen~show()
  if ret == .CommonItemDialog~canceled then say 'The user canceled'
  else say 'The user picked:' fileOpen~getResult

::requires 'ooDialog.cls'

::class 'CDEvents' subclass CommonDialogEvents

::method onFolderChanging unguarded
  expose labelChanged
  use arg cfd, hwnd

  -- Only change the label on the first event.
  if \ labelChanged~dataType('O') then labelChanged = .false

  if \ labelChanged then do
      cfd~setCancelButtonLabel('Abort')
      labelChanged = .true
  end

  return self~S_OK
```

32.7.18. setClientGuid

```
>>--setClientGuid(--guid--)----------------------><
```

Allows the calling application to associate a GUID with a dialog's persistent state.

Arguments:

The single argument is:

guid [required]

The GUID to assoicate with the dialog's persistent state. A GUID (Globally Unique Identifier) is a unique reference number used as an identifier in computer software. GUID typically refers to an implementations of the Universally Unique Identifier (UUID) standard. The *getGUID* of the *DlgUtil* class can be used to obtain a GUID. Note that the *guid* is specified as a string in conventional format.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

A dialog's persistent state can include factors such as the last visited directory and the size and position of the dialog. Typically, this state is persisted based on the name of the executable file.

By specifying a GUID, a program can have different persisted states for different versions of the dialog within the same program, for example, a backup to file dialog and an open text document dialog.

The *setClientGuid* should be called immediately after the instantiation of the dialog object.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example has several GUID strings used for different types of the common item dialog used in an accounting program. The GUID strings have been generated and then assigned to constant values in the program:

```
::class 'EasyAccounts' subclass RcDialog

-- Do not copy these GUIDs into your own code.  Always generate your own GUID.
::constant GUID_OPEN          '021b0a3d-cb16-4def-ab42-7fc259fd84d9'
::constant GUID_SAVE          'beff4495-fa60-468b-90c3-1dcb6fde0e78'
::constant GUID_OPEN_MULTI    'b6c25d80-198e-4baa-aad8-1249e4f15b3f'
::constant GUID_OPEN_CUSTOM   '745c45cb-3464-46fc-96cf-8628ad4bf63b'
::constant GUID_SAVE_DEFAULT  'fe2c5305-d7f9-43cd-9d7b-9c3d4d1b79be'
::constant GUID_OPEN_FOLDER   'ce5ad183-b42f-4f37-a5d6-78d1cffea2e6'

...


  ofd = .OpenFileDialog~new
  ret = ofd~setClientGuid(self~GUID_OPEN)
```

32.7.19. setDefaultExtension

```
>>--setDefaultExtension(--extension--)------------><
```

Sets the default extension to be added to file names.

Arguments:

The single argument is:

extension [required]

The defualt extension for the common item dialog. Do not include the period. *jpg* is correct, while *.jpg* is not.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The extension must be less than 260 characters in length.

If this method is invoked before showing the dialog, the dialog will update the default extension automatically when the user chooses a new file type. See *setFileTypes*.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example sets the default extension to *.cls*. Notice that the *dot* is left off of the string passed to the method, *cls* not *.cls*:

```
 sfd = .SaveFileDialog~new
 ret = sfd~setClientGuid(self~GUID_SAVE_DEFAULT)

 filter = .array~of('Rexx Program File', '*.rex', 'Rexx Class File', '*.cls', 'Text
File', '*.txt')
 sfd~setFileTypes(filter)

 sfd~setFileTypeIndex(2)
 sfd~setDefaultExtension('cls')

 ret = sfd~setSaveAsItem(rexx_home'\ooDialog.cls')

 -- Show the dialog and get the response
 ret = sfd~show(self)
```

32.7.20. setDefaultFolder

```
>>--setDefaultFolder(--folder--)-----------------><
```

Sets the folder used as a default if there is not a recently used folder value available.

Arguments:

The single argument is:

folder [required]

The folder that is set as the default folder when the dialog opens. This can be specified as a full path name, a *CSIDL_XX* name, or an *item ID list*.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

Contrast this method with the *setFolder* method. The dialog will open with the folder specified with *setDefaultFolder* as the selected folder only if there is not a recently used folder saved by the operating system. Whereas the *setFolder* method will explicitly open the dialog with the specified folder as the initial selected folder.

Currently the only way to get an item ID list is through the *getItemIDList* method of the *BrowseForFolder* class. Future enhancements of ooDialog may make more use of item ID list, which is why the *setDefaultFolder* method accepts an item ID list for the *folder* argument.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example uses the *setClientGuid* method so that the state of the open file dialog will be persisted. The very first time the application is used, there will be no saved state, so the application sets the default folder to the ooRexx installation directory, (unless the ooRexx directory can't be found.) The first time the application uses the open file dialog, it will open with the ooRexx installation directory as the initial directory. After that the dialog will open with the last visited directory as the initial directory:

```
guid = '95e626ab-a59d-4779-b7ac-15d49a900ee7'

ofd = .OpenFileDialog~new
ret = ofd~setClientGuid(guid)

initFolder = value('REXX_HOME', , 'ENVIRONMENT')
if initFolder == '' then initFolder = 'C:\Program Files'
ofd~setDefaultFolder(initFolder)
...
```

32.7.21. setFileName

```
>>--setFileName(--fileName--)--------------------><
```

Sets the file name initially placed in the file name edit box of the dialog

Arguments:

The single argument is:

fileName [required]

The initial file name displayed in the edit box.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The *fileName* argument must be less than 260 characters in length.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

32.7.22. setFileNameLabel

```
>>--setFileNameLabel(--label--)------------------><
```

Sets the text of the label next to the file name edit box

Arguments:

The single argument is:

label [required]

The text for the label of the file name edit box.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The *label* argument must be less than 260 characters in length.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example uses the *setFileNameLabel* argument to change the default lable for the file name edit box:

```
ofd = .OpenFileDialog~new
ret = ofd~setClientGuid(self~GUID_OPEN_CUSTOM)

ofd~options = ofd~options~delWord(ofd~options~wordPos('FILEMUSTEXIST'), 1)

ofd~setTitle("The ooRexx Project's Open File Dialog")
ofd~setFileNameLabel('Type in, or select an existing, file to open here:')
ofd~setOkButtonLabel('Finished')

opts = 'FORCEPREVIEWPANEON HIDEMRUPLACES HIDEPINNEDPLACES'
ofd~options = ofd~options opts

-- Okay, we are all set, show the dialog and get the response from the user:
ret = ofd~show(self)
```

32.7.23. setFileTypeIndex

```
>>--setFileTypeIndex(--index--)------------------><
```

Sets the file type that appears as selected in the dialog.

Arguments:

The single argument is:

index [required]

The 1-based index in the file types filter array (*setFileTypes*) that should be the selected file type when the dialog is shown.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The operating system seems to set the file type to that closet to the index number. If index is 0, then file type 1 in the array is selected. In an array of 4 file types, if index is 566, the file type 4 is selected.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example uses the *setFileTypeIndex* method to set the selected file type to the second file type in the file types array. Note that each file type in the array is a pair, that is the first file type is the *Rexx Program File* / **.rex* pair:

```
 sfd = .SaveFileDialog~new
 ret = sfd~setClientGuid(self~GUID_SAVE_DEFAULT)

 filter = .array~of('Rexx Program File', '*.rex', 'Rexx Class File', '*.cls', 'Text
File', '*.txt')
 sfd~setFileTypes(filter)

 sfd~setFileTypeIndex(2)
 sfd~setDefaultExtension('cls')

 ret = sfd~setSaveAsItem(rexx_home'\ooDialog.cls')

 -- Show the dialog and get the response
 ret = sfd~show(self)
```

32.7.24. setFileTypes

```
>>--setFileTypes(--fileTypes--)------------------><
```

Sets the file types filter.

Arguments:

The single argument is:

fileTypes [required]

An array that specifies the file types. Index 1 is friendly name of the filter, index 2 is the filter pattern. Index 3 is the friendly name of the second file type, index 4 is the pattern for the second file type, etc..

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

When using the Open file dialog, the file types declared are used to filter the view. When using the Save file dialog, these values determine which file extension is appended to the file name.

The types array must not be sparse and it must contain an even number of items.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example uses the *setFileTypes* method to set 3 file types for the save file dialog. Note that each file type in the array is a pair, that is the first file type is the *Rexx Program File* / **.rex* pair, the second file type is the *Rexx Class File* / **.cls* pair, etc.:

```
 sfd = .SaveFileDialog~new
 ret = sfd~setClientGuid(self~GUID_SAVE_DEFAULT)
 filter = .array~of('Rexx Program File', '*.rex', 'Rexx Class File', '*.cls', 'Text
File', '*.txt')
 sfd~setFileTypes(filter)

 sfd~setFileTypeIndex(2)
 sfd~setDefaultExtension('cls')

 ret = sfd~setSaveAsItem(rexx_home'\ooDialog.cls')

 -- Show the dialog and get the response from the user
 ret = sfd~show(self)
```

32.7.25. setFilter

```
>>--setFilter(--filter--)------------------------><
```

Sets a special filter that can be used to remove some items from the dialogs view.

Arguments:

The single argument is:

filter [required]

The *ShellItemFilter* object used to do the filtering.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The filter can only be set one time and can not be removed. The filter object can not be reused.

The MSDN *documentation* MSDN says both:

- To filter by file type, *setFileTypes* should be used, because in folders with a large number of items it may offer better performance than applying a *ShellItemFilter*
- Deprecated. *setFilter* method is no longer available for use as of Windows 7

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

In the example programs distributed with ooDialog, there is an example: **`saveFileWithFilter.rex`** that demonstrates how to implement a *ShellItemFilter* subclass and use it with the *setFilter* method.

32.7.26. setFolder

```
>>--setFolder(--folder--)---------------------------------------------><
```

Sets the folder that is selected when the dialog is opened.

Arguments:

The single argument is:

folder [required]

The folder that is set as the selected folder when the dialog opens. This can be specified as a full path name, a *CSIDL_XX* name, or an *item ID list*.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

Contrast this method with the *setDefaultFolder* method. The dialog will open with the folder specified with *setDefaultFolder* as the selected folder only if there is not a recently used folder saved by the operating system. Whereas the *setFolder* method will explicitly open the dialog with the specified folder as the initial selected folder.

Currently the only way to get an item ID list is through the *getItemIDList* method of the *BrowseForFolder* class. Future enhancements of ooDialog may make more use of item ID list, which is why the *setFolder* method accepts an item ID list for the *folder* argument.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

Normally the operating system will pick the initial directory that is selected when the open file dialog is first shown. Usually this is the last directory the user had selected. In this example the *setFolder* method over-rides thats behaviour and forces the dialog to open with the specified directory selected:

```
::method onSaveFile private
  expose rexx_home

  ofd = .OpenFileDialog~new
  ret = ofd~setClientGuid(self~GUID_OPEN_FOLDER)

  ofd~setFolder(rexx_home)

  -- Show the dialog and get the response
  selection = ofd~show(self)
  return selection
```

32.7.27. setOkButtonLabel

```
>>--setOkButtonLabel(--label--)------------------><
```

Sets the label of the Ok button in the dialog.

Arguments:

The single argument is:

label [required]

The new label for the Ok button.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The label must be less than 260 characters in length.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example uses the *setOkButtonLabel* to change the label of the Ok button to *Finished*:

```
ofd = .OpenFileDialog~new
ret = ofd~setClientGuid(self~GUID_OPEN_CUSTOM)

ofd~options = ofd~options~delWord(ofd~options~wordPos('FILEMUSTEXIST'), 1)

ofd~setTitle("The ooRexx Project's Open File Dialog")
ofd~setFileNameLabel('Type in, or select an existing, file to open here:')
ofd~setOkButtonLabel('Finished')
```

32.7.28. setTitle

```
>>--setTitle(--title--)--------------------------><
```

Sets the title of the common item dialog.

Arguments:

The single argument is:

title [required]

The title for the common item dialog.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The title must be less than 260 characters in length.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example uses the *setTitle* method to change the title of the open file dialog to *The ooRexx Project's Open File Dialog*

```
ofd = .OpenFileDialog~new
ret = ofd~setClientGuid(self~GUID_OPEN_CUSTOM)

ofd~options = ofd~options~delWord(ofd~options~wordPos('FILEMUSTEXIST'), 1)
```

```
ofd~setTitle("The ooRexx Project's Open File Dialog")
ofd~setFileNameLabel('Type in, or select an existing, file to open here:')
ofd~setOkButtonLabel('Finished')
```

32.7.29. show

```
>>--show(--+---------+--)------------------------><
           +--owner--+
```

Launches the modal dialog window.

Arguments:

The single argument is:

owner [optional]

The Rexx dialog object that is the owner of the common item dialog. The owner dialog is disabled until the *show* method returns, which is usually the desired behavior.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The return from the *show* method specifies if the dialog was shown with out error, and or if the user canceled the dialog. S_OK indicates that the dialog was shown without error and the user made a selection. If the user cancels the dialog the return is the *ComConstants* CANCELED value. Any other value is an error code. To actually get the selection, the programmer uses the *getResult* or the *getResults* methods.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This code snippet is executed from within a method of an ooDialog dialog subclass. It sets the subclass object as the owner of the open file dialog.

```
ret = ofd~show(self)

if ret == ofd~canceled then text = 'The user canceled the open'
else text = 'Open file:' ofd~getResult

ofd~release
```

32.7.30. unadvise

```
>>--unadvise-------------------------------------><
```

Removes the event handler that was attached through the *advise* method.

Arguments:

There are no arguments for this method.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

There can only be one *CommonDialogEvents* object assigned to a common item dialog. To stop receiving event notifications an attached event handler can be removed. Once an event handler object has been attached and then removed through the *unadvise* method, that object is no longer valid. The removed object can not be reattached though the *advise* method. Rather a new **`CommonDialogEvents`** object would need to be instantiated and attached.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

32.8. CommonDialogEvents Class

A **`CommonDialogEvents`** object is a collection of event handling methods. When an instantiated common dialog events object is attached to a common item dialog through its *advise* method, the dialog sends event notifications to the common dialog events object.

The **`CommonDialogEvents`** class has an event handler, with a default implementation, for every event notification that the common item dialog can send. To use the common dialog events class, the programmer defines a subclass, decides what notifications she is interested in, and over-rides the event handler method(s) for those notifications.

32.8.1. Method Table

The following table lists the class and instance methods of the **`CommonDialogEvents`** class:

Table 32.6. CommonDialogEvents Class Method Reference

Method	Description
Constant Methods	Lists and explains the *constant* values provided by the **`CommonDialogEvents`** class.
Class Methods	**Class Methods**
new	Instantiates a new **`CommonDialogEvents`** object.
Instance Methods	**Instance Methods**
onButtonClicked	Invoked when the user clicks a push button.
onCheckButtonToggled	Invoked when the user changes the state of a check box.
onControlActivating	Called when an Open button drop-down list customized through the *enableOpenDropDown* method, or a menu is about to display its contents.
onFileOk	Invoked just before the dialog is about to return with a result.
onFolderChange	Invoked when the user navigates to a new folder. This notification is sent after the folder has been changed.
onFolderChanging	Invoked before *onFolderChange*. This notification allows the programmer to stop navigation to a particular location.
onHelp	An *onHelp* event handler was in a MSDN example for common dialog events. But it is not documented and doesn't seem to get invoked. It has been left as a place holder only, do not expect it to work.

Method	Description
onItemSelected	Invoked when an item is selected in a combo box, when a user clicks a radio button (called an option button by the common item dialog documentation), or an item is chosen from a menu.
onItemSelected	Invoked when an item is selected in a combo box, when a user clicks a radio button (called an option button by the common item dialog documentation), or an item is chosen from a menu.
onOverwrite	Invoked from the save dialog when the user chooses to overwrite a file.
onSelectionChange	Invoked when the user changes the selection in the dialog's view.
onShareViolation	Implementing this event handler allows an application to respond to sharing violations that arise from Open or Save operations.
onTypeChange	Invoked when the dialog is opened to notify the application of the initial chosen filetype.

32.8.2. Constant Methods

The **`CommonDialogEvents`** class provides a number of *constant* values through the use of the **`::constant`** directive. The constants are listed and documented in this section. Recall that the constant methods defined by the **`::constant`** directive are both class and instance methods of the class they are defined in.

The three **`FDESVR_x`** constants are used to indicate an application's response to a sharing violation that occurs when a file is opened or saved. They are the valid reply values for the *onShareViolation* or *onOverwrite* events.

The constants provided by the **`CommonDialogEvents`** are listed in the table below:

Table 32.7. CommonDialogEvents Constant Reference

Constant Symbol	Description
FDESVR_ACCEPT	The application has determined that the file should be returned from the common item dialog.
FDESVR_DEFAULT	The application has not handled the event. The common item dialog displays a user interface (UI) indicating that the file is in use and a different file must be chosen.
FDESVR_REFUSE	The application has determined that the file should not be returned from the common item dialog.

32.8.3. new (Class Method)

```
>>--new------------------------------------------><
```

Instantiates a new **`CommonDialogEvents`** object.

Arguments:

There are no arguments to the *new* method.

Return value:

A newly instantiated **`CommonDialogEvents`** object.

Details:

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example instantiates a common dialog events object and attaches it to the open file dialog through the *advise* method:

```
ofd = .OpenFileDialog~new
ev = .CDEvents~new
ofd~advise(ev)
```

32.8.4. onButtonClicked

```
::method onButtonClicked unguarded
    use arg cfd, hwnd, id

    return self~S_OK
```

Invoked when the user clicks a push button.

Arguments:

The arguments are:

cfd

The Rexx *CommonItemDialog* object that is sending the event notification.

hwnd

The window *handle* of the underlying Windows common item dialog.

id

The resource ID of the button that was clicked

Return value:

The return from the event handler seems to be ignored by the operating system. The programmer should return S_OK.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example comes from an application that has a lot of customizations allowing the user to enter a lot of metadata that is saved by the application along with saving a file. The user clicks a push button to have the application verify that all the required metadata has been entered ...

```
::class 'CDEvents' subclass CommonDialogEvents

::method onButtonClicked unguarded
  use arg cfd, hwnd, item

  if item == .constDir[IDC_PB_VERIFY] then do
    text = cfd~getEditBoxText(IDC_EDIT)
    if text == '' then do
      title = 'Save File Error'
      msg = 'You must enter a comment when saving this'.endOfLine || -
```

```
              'file. The comment will be saved along with the file.'

        j = MessageDialog(msg, hwnd, title, OK, ERROR)
      end
    end
    return self~S_OK
```

32.8.5. onCheckButtonToggled

```
::method onCheckButtonToggled unguarded
  use arg cfd, hwnd, id, checked

  return self~S_OK
```

Invoked when the user changes the state of a check box.

Arguments:

The arguments are:

cfd
: The Rexx *CommonItemDialog* object that is sending the event notification.

hwnd
: The window *handle* of the underlying Windows common item dialog.

id
: The resource ID of the check box that was toggled.

checked
: True if the check box is now checked, false if it is now unchecked.

Return value:

The return from the event handler seems to be ignored by the operating system. The programmer should return S_OK.

Details

Raises syntax errors when incorrect usage is detected.

32.8.6. onControlActivating

```
::method onControlActivating unguarded
  use arg cfd, hwnd, id

  return self~S_OK
```

Called when an Open button drop-down list customized through the *enableOpenDropDown* method, or a menu is about to display its contents.

Arguments:

The arguments are:

cfd
: The Rexx *CommonItemDialog* object that is sending the event notification.

hwnd

The window *handle* of the underlying Windows common item dialog.

id

The resource ID of the drop-down list or tool menu that is about to be displayed.

Return value:

The return from the event handler seems to be ignored by the operating system. The programmer should return S_OK.

Remarks:

In response to this notification, an application can update the contents of the menu or list about to be displayed, based on the current state of the dialog.

Details

Raises syntax errors when incorrect usage is detected.

32.8.7. onFileOk

```
::method onFileOk unguarded
  use arg cfd, hwnd

  return self~S_OK
```

Invoked just before the dialog is about to return with a result.

Arguments:

The arguments are:

cfd

The Rexx *CommonItemDialog* object that is sending the event notification.

hwnd

The window *handle* of the underlying Windows common item dialog.

Return value:

Return S_OK to accept the current result in the dialog or S_FALSE to refuse it. When S_FALSE is returned, the dialog will remain open.

Remarks:

When this event handler is invoked, the *getResult* and *getResults* methods can be invoked. The application can use this event handling method to perform additional validation before the dialog closes, or to prevent the dialog from closing. If the application prevents the dialog from closing, it should display something to the user to indicate why the file is not accepted. Typically the *hwnd* argument would be used as the owner window of the message dialog.

An application can also use this method to perform all of its work surrounding the opening or saving of files.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example comes from a business application that does not allow the user to save or open files that are marked as *reserved*. Reserved files are marked by the administrator by placing the string *resv* in the file name:

```
::method onFileOk unguarded
    use arg ofd, hwnd

    fileName = ofd~getCurrentSelection
    if fileName~caselessPos('resv') <> 0 then do
        msg = '"Reserved" files can not be opened.  Please select another file.'
        title = 'File Open Error'
        r = MessageDialog(msg, hwnd, title, 'OK', 'WARNING')
        return self~S_FALSE
    end

    return self~S_OK
```

32.8.8. onFolderChange

```
::method onFolderChange unguarded
  use arg cfd, hwnd

  return self~S_OK
```

Invoked when the user navigates to a new folder. This notification is sent after the folder has been changed.

Arguments:

The arguments are:

cfd

The Rexx *CommonItemDialog* object that is sending the event notification.

hwnd

The window *handle* of the underlying Windows common item dialog.

Return value:

The return from the event handler seems to be ignored by the operating system. The programmer should return S_OK.

Remarks:

This notification is sent when the dialog is opened.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example ...

```
  .constDir[IDC_TEXT_DIR]   = 300
  .constDir[IDC_TEXT_LABEL] = 305

  .CommonItemDialog~inherit(.CommonDialogCustomizations)
  fileOpen = .OpenFileDialog~new
  events = .CDEvents~new
  fileOpen~advise(events)

  ret = fileOpen~startVisualGroup(IDC_TEXT_LABEL, 'Current Folder:')
  ret = fileOpen~addText(IDC_TEXT_DIR, "")
  ret = fileOpen~endVisualGroup()

  filter = .array~of('Rexx Program', '*.rex', 'Text Document', '*.txt', 'Hidden Console
 Rexx Program', '*.rxg')
  fileOpen~setFileTypes(filter)
  fileOpen~setFileTypeIndex(1)

  if fileOpen~show() == .CommonItemDialog~canceled then
      say 'The user canceled'
  else
      say 'The user picked:' fileOpen~getResult

  fileOpen~release

::requires 'ooDialog.cls'

::class 'CDEvents' subclass CommonDialogEvents

::method onFolderChange unguarded
  use arg fileOpen, hwnd

  folder = fileOpen~getFolder
  fileOpen~setControlLabel(IDC_TEXT_DIR, folder)
  return self~S_OK
```

32.8.9. onFolderChanging

```
::method onFolderChanging unguarded
  use arg cfd, hwnd, folder

  return self~S_OK
```

Invoked before *onFolderChange*. This notification allows the programmer to stop navigation to a particular location.

Arguments:

The arguments are:

cfd

The Rexx *CommonItemDialog* object that is sending the event notification.

hwnd

The window *handle* of the underlying Windows common item dialog.

folder

The complete path name of the folder the user is about to change to.

Return value:

Return S_OK or E_NOTIMPL to allow the folder change to occur. Return S_FALSE to prevent the folder change.

Remarks:

The programmer can call *setFolder* from within this event handler to redirect navigation to an alternate folder. The actual navigation does not occur until *onFolderChanging* has returned.

If the application simply prevents navigation to a particular folder, it should display something to the user to indicate why the folder change is restricted. Typically the *hwnd* argument would be used as the owner window of the message dialog.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This is a contrived example used to show the basics of the *onFolderChanging* event handler. The user is prevented from navigating to any folder that contains *rexx* in the folder name:

```
  .CommonItemDialog~inherit(.CommonDialogCustomizations)
  fileOpen = .OpenFileDialog~new

  events = .CDEvents~new
  fileOpen~advise(events)
  fileOpen~setFolder('C:\Program Files')

  if fileOpen~show() == .CommonItemDialog~canceled then
      say 'The user canceled'
  else
      say 'The user picked:' fileOpen~getResult

  fileOpen~release

::requires 'ooDialog.cls'

::class 'CDEvents' subclass CommonDialogEvents

::method onFolderChanging unguarded
  use arg fileOpen, hwnd, folder

  if folder~caselessPos('Rexx') <> 0 then do
      msg = 'Navigating to a "Rexx" folder is not allowed.  Please select another
 folder.'
      title = 'Folder Navigation Error'
      r = MessageDialog(msg, hwnd, title, 'OK', 'WARNING')
      return self~S_FALSE
  end

  return self~S_OK
```

32.8.10. onHelp

```
::method onHelp unguarded
  use arg cfd, hwnd

  return self~S_OK
```

An *onHelp* event handler was in a MSDN example for common dialog events. But it is not documented and doesn't seem to get invoked. It has been left as a place holder only, do not expect it to work.

32.8.11. onItemSelected

```
::method onItemSelected unguarded
  use arg cfd, hwnd, itemID, containerID
  return self~S_OK
```

Invoked when an item is selected in a combo box, when a user clicks a radio button (called an option button by the common item dialog documentation), or an item is chosen from a menu.

Arguments:

The arguments are:

cfd
: The Rexx *CommonItemDialog* object that is sending the event notification.

hwnd
: The window *handle* of the underlying Windows common item dialog.

itemID
: The resource ID of the selected item in the container control.

containerID
: The resource ID of the control that contains the selected item.

Return value:

The return from the event handler seems to be ignored by the operating system. The programmer should return S_OK.

Remarks:

This notification is not sent when the user chooses an item from the drop-down menu attached to the Open button. MSDN gives this explanation for that behaviour: *the action taken in that case is always the same: close the dialog as if the user had simply clicked the Open button.* In this situation, the programmer can call the *getSelectedControlItem* to obtain the item the user chose from the menu.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example checks the appropriate radio button when the user selects a menu item:

```
::class 'CDEvents' subclass CommonDialogEvents

::method onItemSelected
  use arg cfd, hwnd, item, container

  if container == .constDir[IDC_MENU] then do
    select
      when item == .constDir[IDC_MENU_REX] then do
          cfd~setSelectedControlItem(IDC_RBL_RADIOS, IDC_RB_REX)
          cfd~setFileTypeIndex(1)
      end
      when item == .constDir[IDC_MENU_TXT] then do
          cfd~setSelectedControlItem(IDC_RBL_RADIOS, IDC_RB_TXT)
          cfd~setFileTypeIndex(2)
      end
      when item == .constDir[IDC_MENU_CONSOLE] then do
          cfd~setSelectedControlItem(IDC_RBL_RADIOS, IDC_RB_RXG)
          cfd~setFileTypeIndex(3)
      end
      otherwise nop
    end
    -- End select
  end
  return self~S_OK
```

32.8.12. onOverwrite

```
::method onOverwrite unguarded
  use arg cfd, hwnd, fileName

  return self~E_NOTIMPL
```

Invoked from the save dialog when the user chooses to overwrite a file.

Arguments:

The arguments are:

cfd
: The Rexx *CommonItemDialog* object that is sending the event notification.

hwnd
: The window *handle* of the underlying Windows common item dialog.

fileName
: The complete path name of the file that will be overwritten.

Return value:

The event handler should return one of the *FDESVR_xx* constants or E_NOTIMPL. Returning any other value may cause unpredictable behavior.

Remarks:

This event handler is only invoked if the common item dialog has the *OVERWRITEPROMPT* option set.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example comes from a business application where IT marks certain files that can not be overwritten. However, when the user tries to overwrite a restricted file, it is possible to overwrite the file by entering the proper key code:

```
  ::class 'CDEvents' subclass CommonDialogEvents

::method onOverwrite unguarded
  use arg cfd, hwnd, file

  magicText = cfd~getEditBoxText(IDC_EDIT)
  fileName  = fileSpec('N', file)

  if fileName~caselessPos('reserv') <> 0 then do
    title = 'IT File Overwrite Manager'

    if magicText == self~magicKey then do
      msg = "Permission key matches.  Permission to overwrite" file 'has been granted.'
      j = MessageDialog(msg, hwnd, title, 'OK', 'INFORMATION')
      return self~FDESVR_ACCEPT
    end
    else if magicText == '' then do
      msg = "IT does not allow overwriting this file.  Permission to overwrite " || -
            "the file can be granted by entering the correct key in the edit box."
      j = MessageDialog(msg, hwnd, title, 'OK', 'WARNING')
      return self~FDESVR_REFUSE
    end
    else do
      msg = "IT does not allow overwriting this file.  Overwrite permission denied."
      j = MessageDialog(msg, hwnd, title, 'OK', 'ERROR')
      return self~FDESVR_REFUSE
    end
  end
  return self~FDESVR_ACCEPT
```

32.8.13. onSelectionChange

```
::method onSelectionChange unguarded
  use arg cfd, hwnd

  return self~S_OK
```

Invoked when the user changes the selection in the dialog's view.

Arguments:

The arguments are:

cfd

The Rexx *CommonItemDialog* object that is sending the event notification.

hwnd

The window *handle* of the underlying Windows common item dialog.

Return value:

The return from the event handler seems to be ignored by the operating system. The programmer should return S_OK.

Remarks:

Testing shows that the common item dialog sends this notification when the selecte file changes and when the selected folder changes

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example shows a simple event handler that can be used to test when the dialog sends the selection change notification:

```
::method onSelectionChange unguarded
  use arg fileOpen, hwnd
  say 'Got onSelectionChange() notification. Selection is:' fileOpen~getCurrentSelection

  return self~S_OK
```

32.8.14. onShareViolation

```
::method onShareViolation unguarded
  use arg cfd, hwnd, fileName

  return self~E_NOTIMPL
```

Implementing this event handler allows an application to respond to sharing violations that arise from Open or Save operations.

Arguments:

The arguments are:

cfd

The Rexx *CommonItemDialog* object that is sending the event notification.

hwnd

The window *handle* of the underlying Windows common item dialog.

fileName

The complete path name of the file that will be overwritten.

Return value:

The event handler should return one of the *FDESVR_xx* constants or E_NOTIMPL. Returning any other value may cause unpredictable behavior.

Remarks:

A sharing violation can occur if an application tries to read or modify a file when another process holds the file lock for that file.

This event handler is only invoked if the common item dialog has the *SHAREAWARE* option set.

Details

Raises syntax errors when incorrect usage is detected.

32.8.15. onTypeChange

```
::method onTypeChange unguarded
  use arg cfd, hwnd

  return self~S_OK
```

Invoked when the dialog is opened to notify the application of the initial chosen filetype.

Arguments:

The arguments are:

cfd

The Rexx *CommonItemDialog* object that is sending the event notification.

hwnd

The window *handle* of the underlying Windows common item dialog.

Return value:

The return from the event handler seems to be ignored by the operating system. The programmer should return S_OK.

Remarks:

onTypeChange would make a good event handler for the purpose of performing some action when the common file dialog is about to be opened, but has not been shown yet. However the notification is only sent if the application sets a file types (*setFileTypes*) filter.

Details

Raises syntax errors when incorrect usage is detected.

Example:

This example changes the label of the Cancel button in a file open dialog. That label can only be changed after the dialog is shown and typically a programmer would prefer to do it before the dialog is visible:

```
  fileOpen = .OpenFileDialog~new

  events = .CDEvents~new
  fileOpen~advise(events)

  fileOpen~setOkButtonLabel('Open This File')

  filter = .array~of('Rexx Program', '*.rex', 'Text Document', '*.txt', 'Hidden Console
 Rexx Program', '*.rxg')
  fileOpen~setFileTypes(filter)
  fileOpen~setFileTypeIndex(1)

  if fileOpen~show() == .CommonItemDialog~canceled then
      say 'The user canceled'
  else
      say 'The user picked:' fileOpen~getResult

  fileOpen~release

::requires 'ooDialog.cls'

::class 'CDEvents' subclass CommonDialogEvents

::method onTypeChange unguarded
  expose labelChanged
  use arg cfd, hwnd

  -- Only change the label on the first event.
  if \ labelChanged~dataType('O') then
      labelChanged = .false

  if \ labelChanged then do
      cfd~setCancelButtonLabel('Abort')
      labelChanged = .true
  end

  return self~S_OK
```

32.9. OpenFileDialog Class

The OpenFileDialog class is a concrete subclass of the *CommonItemDialog*. The programmer does not instantiate a new **`CommonItemDialog`** object, but rather instantiates a new **`OpenFileDiaog`** or *SaveFileDialog* object. The **`OpenFileDialog`** is only available on Windows Vista or later operating systems.

32.9.1. Method Table

The following table lists the class and instance methods of the **`OpenFileDialog`** class. Most of the methods come from the Common Item Dialog and are listed here for convenience. The method name(s) in bold in the table are unique to the open file dialog:

Table 32.8. OpenFileDialog Class Method Reference

Method	Description
Class Methods	**Class Methods**
new	Instantiates a new open file dialog object
Instance Methods	**Instance Methods**
addPlace	Adds a folder to the list of places available for the user to open or save items.
advise	Assigns an event handler object that invokes methods in the ooDialog program for event notificationss coming from the common item dialog.
clearClientData	Instructs the file dialog to clear all persistent state information.
close	Closes the dialog..
getCurrentSelection	Gets the user's current selection in the dialog.
getFileName	Retrieves the text currently entered in the dialog's File name edit box.
getFileTypeIndex	Gets the one-based index of the currently selected file type.
getFolder	Retrieves either the folder currently selected in the dialog, or, if the dialog is not currently displayed, the folder that is to be selected when the dialog is opened.
getResult	Gets the choice that the user made in the dialog.
getResults	Gets the files selected by the user when multi-file selection is enabled.
initCOM	Initializes the COM *library* on the current thread.
isReleased	Tests if the file dialog object has been *released*.
release	Releases the system resources used by the underlying COM object that this Rexx object represents.
releaseCOM	Releases the COM library on the current thread.
setCancelButtonLabel	Sets the label of the Cancel button in the dialog.
setClientGuid	Allows the calling application to associate a GUID with a dialog's persistent state.
setDefaultExtension	Sets the default extension to be added to file names.
setDefaultFolder	Sets the folder used as a default if there is not a recently used folder value available.
setFileName	Sets the file name initially placed in the file name edit box of the dialog
setFileNameLabel	Sets the text of the label next to the file name edit box
setFileTypeIndex	Sets the file type that appears as selected in the dialog.
setFileTypes	Sets the file types filter.
setFilter	Sets a special filter that can be used to remove some items from the dialog's view.
setFolder	Sets the folder that is selected when the dialog is opened.
setOkButtonLabel	Sets the label of the Ok button in the dialog.

Method	Description
setTitle	Sets the title of the common item dialog.
show	Launches the modal dialog window.
unadvise	Removes the event handler that was attached through the *advise* method.

32.9.2. new (Class Method)

```
>>--new-----------------------------------------><
```

Instantiates a new open file dialog.

Arguments:

There are new arguments to this method.

Return value:

Returns a new **`OpenFileDialog`** object.

Remarks:

It is highly unlikely that anything will fail when instantiating a new open file dialog. However, if there is an error, the **`.SystemErrorCode`** will be set to an error code and the *isReleased* method will return true

Details:

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example instantiates a new open file dialog and checks for error. Typically the author does not perform this error check, but that is not to suggest it shouldn't be done:

```
ofd = .OpenFileDialog~new
if ofd~isReleased | .SystemErrorCode <> ofd~S_OK then return 99
...
```

32.9.3. getResults

```
>>--getResults(--+--------+--)---------------------------------------------><
                 +--sigdn--+
```

Gets the files selected by the user when multi selection is enabled.

Arguments:

The single argument is:

sigdn [optional]

A *SIGDN* keyword that specfies the format of the names of the shell items chosen by the user. The default if this arugment is omitted is the FILESYSPATH keyword. If not omitted, the *sigdn* argument specifies the format of the returned names using exactly one of the *SIGDN* keywords. All returned items are formatted the same.

Return value:

On success returns an array of the file names. On error returns the **`.nil`** object.

Remarks:

This method fails when the open file dialog did not have multi selection enabled or if the user canceled the dialog.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example initiates an open file dialog and enables the multi-file selection capability. The dialog is shown and the user's response is formatted into a text message which is used to display the results

```
ofd = .OpenFileDialog~new
ret = ofd~setClientGuid(self~GUID_OPEN_MULTI)

-- To allow the user to select multiple files, we need to add this keyword
-- to the options:
ofd~options = ofd~options 'ALLOWMULTISELECT'

-- Show the dialog and get the response
ret = ofd~show(self)

if ret == ofd~canceled then do
    text = 'The user canceled the open operation.'
end
else do
    files = ofd~getResults
    if files~items == 1 then do
        text = 'Open file:' files[1]
    end
    else do
        text = 'Open files (file names only):'
        msg = 'Open files (complete file names as returned):' || .endOfLine~copies(2)

        do f over files
            text ||= ' ' || fileSpec('N', f)
            msg  ||= f || .endOfLine
        end
    end
end
ofd~release
```

32.10. SaveFileDialog Class

A SaveFileDialog object is a concrete subclass of the *CommonItemDialog*. The programmer does not instantiate a new **`CommonItemDialog`** object, but rather instantiates a new **`SaveFileDialog`** or *OpenFileDialog* object. The **`SaveFileDialog`** is only available on Windows Vista or later operating systems.

32.10.1. Method Table

The following table lists the class and instance methods of the **`SaveFileDialog`** class. Most of the methods come from the Common Item Dialog and are listed here for convenience. The method name(s) in bold in the table are unique to the save file dialog:

Table 32.9. SaveFileDialog Class Method Reference

Method	Description
Class Methods	**Class Methods**
new	Instantiates a new save file dialog object.
Instance Methods	**Instance Methods**
addPlace	Adds a folder to the list of places available for the user to open or save items.
advise	Assigns an event handler object that invokes methods in the ooDialog program for event notificationss coming from the common item dialog.
clearClientData	Instructs the file dialog to clear all persistent state information.
close	Closes the dialog..
getCurrentSelection	Gets the user's current selection in the dialog.
getFileName	Retrieves the text currently entered in the dialog's File name edit box.
getFileTypeIndex	Gets the one-based index of the currently selected file type.
getFolder	Retrieves either the folder currently selected in the dialog, or, if the dialog is not currently displayed, the folder that is to be selected when the dialog is opened.
getResult	Gets the choice that the user made in the dialog.
initCOM	Initializes the COM *library* on the current thread.
isReleased	Tests if the file dialog object has been *released*.
release	Releases the system resources used by the underlying COM object that this Rexx object represents.
releaseCOM	Releases the COM library on the current thread.
setCancelButtonLabel	Sets the label of the Cancel button in the dialog.
setClientGuid	Allows the calling application to associate a GUID with a dialog's persistent state.
setDefaultExtension	Sets the default extension to be added to file names.
setDefaultFolder	Sets the folder used as a default if there is not a recently used folder value available.
setFileName	Sets the file name initially placed in the file name edit box of the dialog
setFileNameLabel	Sets the text of the label next to the file name edit box
setFileTypeIndex	Sets the file type that appears as selected in the dialog.
setFileTypes	Sets the file types filter.
setFilter	Sets a special filter that can be used to remove some items from the dialog's view.
setFolder	Sets the folder that is selected when the dialog is opened.
setOkButtonLabel	Sets the label of the Ok button in the dialog.
setSaveAsItem	Sets the item to be used as the initial entry in the save file dialog.

Method	Description
setTitle	Sets the title of the common item dialog.
show	Launches the modal dialog window.
unadvise	Removes the event handler that was attached through the *advise* method.

32.10.2. new (Class Method)

```
>>--new-----------------------------------------><
```

Instantiates a new save file dialog object.

Arguments:

There are new arguments to this method.

Return value:

Returns a new **SaveFileDialog** object.

Remarks:

It is highly unlikely that anything will fail when instantiating a new save file dialog. However, if there is an error, the **.SystemErrorCode** will be set to an error code and the *isReleased* method will return true.

Details:

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example instantiates a new save file dialog and checks for error. Typically the author does not perform this error check, but that is not to suggest it shouldn't be done:

```
sfd = .SaveFileDialog~new
if sfd~isReleased | .SystemErrorCode <> sfd~S_OK then return 99
...
```

32.10.3. setSaveAsItem

```
>>--setSaveAsItem(--filePathName--)--------------><
```

Sets the item to be used as the initial entry in the save file dialog.

Arguments:

The single argument is:

filePathName [required]

The initial file name. This must be the complete path name of an *existing* file.

Return value:

Returns a *HRESULT code*. S_OK if successful, or an error value otherwise.

Remarks:

The text of the file name edit box is set to the file name part of *filePathName*, and the containing folder is opened in the view of the dialog.

The *setSaveAsItem* method would generally be used when the application is saving a file that already exists. This method fails if the full path name is not used, or if the file does not exist. For files that do not already exist, *setFileName* and *setFolder* methods can be used to place a file name in the edit box and open the view to a specific folder.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example opens a save file dialog and sets the initial save as name to **`ooDialog.cls`** in the ooRexx installation directory:

```
::method saveToDefaultFile unguarded private
    expose rexx_home

    sfd = .SaveFileDialog~new
    ret = sfd~setClientGuid(self~GUID_SAVE_DEFAULT)

    ret = sfd~setSaveAsItem(rexx_home'\ooDialog.cls')
    ...
```

32.11. ShellItemFilter Class

A **`ShellItemFilter`** object can be used if an application needs to perform special filtering to remove some items from the common item dialog's view. The class contains only one instance method, *includeItem*. When a shell item filter object is set as the filter, through the *setFilter* method, of a common item dialog, the *includeItem* method is invoked for each item that would normally be included in the view. If the method returns S_OK the item is included in the view and if the method returns S_FALSE the item is not included in the view.

The default implementation of *includeItem* simply returns S_OK for every item. To use the shell item filter to actually filter the items in the view of a common item dialog, the programmer creates a subclass of the **`ShellItemFilter`** and sets that object as the filter for a common item dialog using the *setFilter* method. In the subclass, the programmer over-rides the *includeItem* method and returns S_OK or S_FALSE for each item, depending on the logic of the application.

The MSDN *documentation* MSDN says both:

- To filter by file type, *setFileTypes* should be used, because in folders with a large number of items it may offer better performance than applying a *ShellItemFilter*
- Deprecated. *setFilter* method is no longer available for use as of Windows 7

32.11.1. Method Table

The following table lists the class and instance methods of the **`ShellItemFilter`** class:

Table 32.10. ShellItemFilter Class Method Reference

Method	Description
Class Methods	**Class Methods**

Method	Description
new	Instantiates a new **`ShellItemFilter`** object.
Instance Methods	**Instance Methods**
includeItem	Sets the status of a specific shell item to be included in the view, or not included.

32.11.2. new (Class Method)

```
>>--new-------------------------------------------><
```

Instantiates a new **`ShellItemFilter`** object.

Arguments:

There are no arguments to the *new* method of the **`ShellItemFilter`** class.

Return value:

Returns a newly instantiated **`ShellItemFilter`** object.

Details:

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example shows the basics of using a shell item filter. The *includeItem* documentation shows the basics of implementing an *includeItem* event handler:

```
  sfd = .SaveFileDialog~new
  filterObj = .SIFilter~new
  sfd~setFilter(filterObj)

  ...

::class 'SIFilter' subclass ShellItemFilter
```

32.11.3. includeItem

```
::method includeItem unguarded
  use arg cfd, dlgHwnd, filePathName

  return self~S_OK
```

The *includeItem* method is in effect an event handler method. It is invoked when the underlying common item dialog COM object is about to add an item to its view. The *includeItem* method handles that event by reply S_OK to allow the item to be added, or by replying S_FALSE to exclude the item form the view.

Arguments:

The arguments are:

cfd [required]

The Rexx *CommonItemDialog* object that is about to add an item to the view.

dlgHwnd [required]

The window *handle* of the common item dialog.

filePathName [required]

The complete path name of the item about to be added.

Return value:

Return S_OK to allow the item to be added to the view. Return S_FALSE to exclude the item from the view.

Remarks:

The default implementation of the *includeItem* returns S_OK for every item. For the **ShllItemFilter** object to be of any use, the programmer must over-ride this method and return S_FALSE for some items to exclude them.

Details

Raises syntax errors when incorrect usage is detected.

Sets the *.SystemErrorCode* variable.

Example:

This example shows an over-ride of the *includeItem* method used to filter out file names that are not to be used as a save file. If any file has the string *reserved*, case insensitive, in its name, it is not shown in the save file dialog's view:

```
::method includeItem
    use arg sfd, hwnd, item

    if item~caselessPos('reserved') <> 0 then return self~S_FALSE

    return self~S_OK
```

32.12. SimpleFolderBrowse Class

The SimpleFolderBrowse object displays the Windows common Browse For Folder dialog and returns the user's selection. It is a simplified version of the *BrowseForFolder* dialog that is designed to be easy to use, and does not have as many configuration options as the **BrowseForFolder** object does.

The Windows common Browse For Folder dialog allows the user to select a folder from a tree-like display of the computer's file system. The Browse For Folder dialog is specifically for opening or choosing folders.

32.12.1. Method Table

The following table lists the class and instance methods of the **SimpleFolderBrowse** class. This simple to use dialog only has a single class method:

Table 32.11. SimpleFolderBrowse Class Method Reference

Method	Description
Class Methods	**Class Methods**
getFolder	Pops up the Browse For Folder Windows dialog and returns the user's selection.

32.12.2. getFolder (Class Method)

```
>>--getFolder(--+-------+-+-------+-+------ +-+---------+-+--------+-+---------+--)--><
                +--capt-+ +-,-bnr-+ +-,-hnt-+ +-,-start-+ +-,-root-+ +-,-owner-+
```

Instantiates the Rexx SimpleBrowseFolder object, pops up the Windows Browse For Folder dialog, and returns the user's selection in one step.

Arguments:

The arguments are:

capt [optional]
: Specifies a caption, or title, for the dialog. The Windows default title is *Browse For Folder*.

bnr [optional]
: Specifies text to use for what Windows calls the *banner* for the dialog. This string is displayed above the tree view control in the dialog box. The string can be used to specify instructions to the user. By default nothing is displayed for the banner and the area above the tree view is blank.

hnt [optional]
: Specifies text to use for a *hint* in the dialog. The hint is displayed below the bottom of the tree view control and above the bottom row of buttons of the dialog. By default there is no hint, and the area where the hint would go is removed from the dialog. That is, the bottom row of buttons is placed directly below the tree view.

start [optional]
: Specifies an initial selected folder for the dialog. By default, the dialog opens with the top-most item in the tree view selected. If a start folder is specified, the folder is selected and, if needed, the tree view items are expanded and the tree view is scrolled so that the starting folder is visible in the tree view.

  Testing indicates that the folder specified must be a fully qualified path name of an existing directory. If not, the operating system seems to just ignore it.

root [optional]
: Specifies the root for the tree view in the dialog. The root is the location of the root folder from which the user starts browsing. Only the specified folder and any subfolders that are beneath it in the namespace hierarchy will appear in the dialog box. The user can not browse above the root folder. If this argument is omitted, the operating system uses the namespace root (the desktop folder.)

  Specify either a fully qualified file system path name, or use one of the *Special Folder Names*.

owner [optional]
: The Rexx dialog object that is to be the owner window of the browse for folder dialog. The owner dialog is disabled until the browse for folder dialog is closed by the user.

Return value:

If the user cancels the dialog, the empty string is returned, otherwise the fully qualified path to the folder the user picked is returned. However it is possible to set up the dialog in a way such that the user can pick a virtual folder. Virtual folders do not have a file system path. The `.nil` object is returned if a virtual folder is picked.

Remarks:

The **`SimpleFolderBrowse`** dialog is intended to be simple to use and is restricted to file system paths. The Windows Browse For Folders dialog has more capabilities than can be accessed through the **`SimpleFolderBrowse`** class, but the capabilities are less likely to be useful in most ooRexx programs. The *BrowseForFolder* class allows access to all features of the Windows Browse For Folders dialog, but it is correspondingly more complex to use.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example puts up the Browse For Folder dialog and gets the user's selection:

```
folder = .SimpleFolderBrowse~getFolder

select
    when folder == .nil then text = 'The user picked a virtual folder'
    when folder == '' then text = 'The user canceled'
    otherwise text = 'The user picked:' folder
end
-- End select

say text
```

32.13. SimpleFolderBrowse Routine

```
>>-SimpleFolderBrowse(-+-------+-+-------+-+-------+-+--------+-+------+-+--------+-)-><
                       +--capt-+ +-,-bnr-+ +-,-hnt-+ +-,-strt-+ +-,-rt-+ +-,-ownr-+
```

Brings up the Windows *Browse For Folder* dialog, allowing the user to select a folder in the file system.

Arguments:

The arguments are:

capt [optional]

Specifies a caption, or title, for the dialog. The Windows default title is *Browse For Folder*.

bnr [optional]

Specifies text to use for what Windows calls the *banner* for the dialog. This string is displayed above the tree view control in the dialog box. The string can be used to specify instructions to the user. By default nothing is displayed for the banner and the area above the tree view is blank.

hnt [optional]

Specifies text to use for a *hint* in the dialog. The hint is displayed below the bottom of the tree view control, and above the bottom row of buttons of the dialog. By default there is no hint, and the area where the hint would go is removed from the dialog. That is, the bottom row of buttons is placed directly below the tree view.

strt [optional]

Specifies an initial selected folder for the dialog. By default, the dialog opens with the top-most item in the tree view selected. If a start folder is specified, the folder is selected and, if needed, the tree view items are expanded and the tree view is scrolled so that the starting folder is visible in the tree view.

Testing indicates that the folder specified must be a fully qualified path name of an existing directory. If not, the operating system seems to just ignore it.

rt [optional]

Specifies the root for the tree view in the dialog. The root is the location of the root folder from which the user starts browsing. Only the specified folder and any subfolders that are beneath it in the namespace hierarchy will appear in the dialog box. The user can not browse above the root folder. If this argument is omitted, the operating system uses the namespace root (the desktop folder.)

Specify either a fully qualified file system path name, or use one of the *Special Folder Names*.

ownr [optional]

The Rexx dialog object that is to be the owner window of the browse for folder dialog. The owner dialog is disabled until the browse for folder dialog is closed by the user.

Return value:

If the user cancels the dialog, the empty string is returned, otherwise the fully qualified path to the folder the user picked is returned. However it is possible to set up the dialog in a way such that the user can pick a virtual folder. Virtual folders do not have a file system path. The **`.nil`** object is returned if a virtual folder is picked.

Remarks:

The *SimpleFolderBrowse* routine is just a shortcut to using the *SimpleFolderBrowse* class. It is intended to be simple to use and is restricted to file system paths. The Windows Browse For Folders dialog has more capabilities than can be accessed through this routine, but the capabilities are less likely to be useful in most ooRexx programs. The *BrowseForFolder* class allows access to all features of the Windows Browse For Folders dialog, but it is correspondingly more complex to use.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example puts up the Browse For Folder dialog and gets the user's selection:

```
folder = SimpleFolderBrowse()

select
    when folder == .nil then text = 'The user picked a virtual folder'
    when folder == '' then text = 'The user canceled'
    otherwise text = 'The user picked:' folder
end
-- End select

say text
```

Appendix A. Deprecated Classes, Methods, Functions, and Routines

Beginning with ooDialog 4.2.0, many methods and some classes are deprecated. All the deprecated classes and methods are listed in this section, along with information on how to replace the deprecated classes and methods. In general, the deprecated *classes* are no longer needed and references to them can simply be removed. Most, but not all, of the deprecated *methods* can simply be replaced with a another method.

It is the *intent* that older programs using deprecated methods will run without having to make changes to the code. This has been fairly well tested, but as with all intentions, there may be some cases where the intent was not satisfied. If such a case is found, it should be brought to the attention of the developers. However, there are some situations where *internal use only* methods have been used, and there are a few deprecated methods that were simply buggy. Programs that no longer run because they used *internal use only* methods are expected to no longer run if the internal implementation is changed. Programs that only worked because of a bug, are also likely to have a problem when the bug is fixed.

There are a number of reasons why classes or methods have been deprecated. Many methods, probably the majority have been deprecated due to the effort to *unify* the naming of methods. The deprecated classes are primarily classes that overcomplicated the programming task, for no good reason. There should be no need to inherit the *AdvancedControls* mixin class in order to use some controls in a dialog, but not have to inherit the class for other controls. Some few methods are deprecated because their implementation is buggy, and a few others are deprecated because they used or relied on programming techniques for Windows 3.1 that no longer work well in a modern Windows version.

In all cases, deprecated methods should never be used in new code. In addition, programmers are strongly encouraged to migrate existing code away from deprecated classes and methods. Deprecated methods usually forward to their replacement methods. Programmers can eliminate this level of redirection by removing the deprecated methods. Deprecated methods and classes are *not* documented in the ooDialog reference, except as listed in this section. If the programmer needs to look up the documentation for a deprecated class or method, he should simply replace that class or method in his code.

A.1. Deprecated Classes

All *deprecated* base and mixin classes are listed in this section. In general, the deprecated classes are simply not needed anymore, and the programmer need do nothing to replace them.

A.1.1. AdvancedControls Mixin Class

This class is *deprecated*. The ooDialog framework no *longer* makes a distinction between some types of dialog controls, and how to use the controls. I.e., the programmer no longer needs to inherit a special class to use some of the dialog controls. All the methods of the AdvancedControls have been moved into the appropriate dialog class. Therefore the programmer does not need to do anything to use any dialog control in any dialog. In new code simply omit inheriting AdvancedControls. In old code, *inherit AdvancedControls* can be deleted from any dialog class declaration and the program will behave the same.

A.1.2. MessageExtensions Mixin Class

This class is *deprecated*. The MessageExtensions class has been renamed to the *EventNotification* class. In addition, the EventNotification class is inherited by the *dialog* object so that the programmer need do nothing special to use event notifications in any dialog. In new code simply omit inheriting MessageExtensions. In old code, *inherit MessageExtensions* can be deleted from any dialog class declaration and the program will behave the same.

A.1.3. PlainUserDialog Class

This class is *deprecated*. The ooDialog framework no *longer* makes a distinction between a PlainUserDialog and an *UserDialog*. Directly subclass the *Userdialog* instead of the *PlainUserDialog*.

A.1.4. CategoryDialog Class

This class is *deprecated*. The **`CategoryDialog`** class was never documented well, was hard for most users to understand, was buggy, and was difficult to debug. In addition, the dialog pages of the **`CategoryDialog`** functioned using a completely different paradigm than all other dialogs. This was not only confusing, but *unnecessary* to begin with.

Use the *PropertySheetDialog*, or a *Tab* control with one of the *Control* dialog classes, to replace the **`CategoryDialog`** class.

A.1.5. PropertySheet Class

This class is *deprecated*. The **`PropertySheet`** class was never documented well, was hard for most users to understand, was buggy, and was difficult to debug. In addition, the dialog pages of the **`PropertySheet`** functioned using a completely different paradigm than all other dialogs. This was not only confusing, but *unnecessary* to begin with.

Use the *PropertySheetDialog*, or a *Tab* control with one of the *Control* dialog classes, to replace the **`PropertySheet`** class.

A.2. Deprecated Dialog Methods

Deprecated dialog methods and their replacement methods are listed in this section.

A.2.1. captureMouse

This method is *deprecated*. Use the *Mouse* object methods directly.

A.2.2. checkMenuItem

This method is *deprecated*. Use the *Menu* object methods directly.

A.2.3. clearButtonRect

This method is *deprecated*. Replace this method with the *clearControlRect* method.

A.2.4. clearMessages

This method is *deprecated*. It does nothing.

A.2.5. combineELwithSB

This method is *deprecated*. It uses an outdated process to add function to a dialog that is similar to a modern up-down control. The method does not work properly. The *UpDown* dialog control should be used directly to replace this method.

A.2.6. connectAnimatedButton

This method is *deprecated*. Replace this method with the *installAnimatedButton* method.

A.2.7. connectBitmapButton

This method is *deprecated*. Replace this method with the *installBitmapbutton* method.

A.2.8. connectEntryLine

This method is *deprecated*. Replace this method with the *connectEdit* method.

A.2.9. connectListControl

This method is *deprecated*. Replace this method with the *connectListView* method.

A.2.10. connectMultiListBox

This method is *deprecated*. Replace this method with the *connectListBox* method.

A.2.11. connectSliderControl

This method is *deprecated*. Replace this method with the *connectTrackBar* method.

A.2.12. connectTabControl

This method is *deprecated*. Replace this method with the *connectTab* method.

A.2.13. connectTreeControl

This method is *deprecated*. Replace this method with the *connectTreeView* method.

A.2.14. deInstall

This method is *deprecated*. It does nothing and always returns 0. In older versions of ooDialog, the *deInstall*() method was used to deregister the external functions with the API manager. This mechanism is no longer needed using the modern APIs introduced in ooRexx 4.0.0.

A.2.15. disableItem

This method is *deprecated*. Replace this method with the *disableControl* method.

A.2.16. disableMenuItem

This method is *deprecated*. Use the *Menu* object methods directly.

A.2.17. dump

This method is *deprecated*. It will simply print: "dump() deprecated." In older versions of ooDialog, the *dump*() method would print out information tied to the undocumented internal implementation of ooDialog. Since the internal implementation has changed, this information no longer exists.

A.2.18. enableItem

This method is *deprecated*. Replace this method with the *enableControl* method.

A.2.19. enableMenuItem

This method is *deprecated*. Use the *Menu* object methods directly.

A.2.20. freeButtonDC

This method is *deprecated*. Replace this method with the *freeControlDC* method.

A.2.21. focusItem

This method is *deprecated*. Replace this method with the *focusControl* method.

A.2.22. getButtonControl

This method is *deprecated*. Replace this method with the *newPushButton* method.

A.2.23. getButtonDC

This method is *deprecated*. Replace this method with the *getControlDC* method.

A.2.24. getButtonRect

This method is *deprecated*. Use the *getControlRect* method instead.

A.2.25. getCheckBox

This method is *deprecated*. Replace this method with the *getCheckBoxData* method.

A.2.26. getCheckControl

This method is *deprecated*. Replace this method with the *newCheckBox* method.

A.2.27. getComboBox

This method is *deprecated*. Replace this method with the *newComboBox* method.

A.2.28. getComboLine

This method is *deprecated*. Replace this method with the *getComboBoxData* method.

A.2.29. getEditControl

This method is *deprecated*. Replace this method with the *newEdit* method.

A.2.30. getEntryLine

This method is *deprecated*. Replace this method with the *getEditData* method.

A.2.31. getGroupBox

This method is *deprecated*. Replace this method with the *newGroupBox* method.

A.2.32. getItem

This method is *deprecated*. Replace this method with the *getControlHandle* method.

A.2.33. getListBox

This method is *deprecated*. Replace this method with the *newListBox* method.

A.2.34. getListControl

This method is *deprecated*. Replace this method with the *newListView* method.

A.2.35. getListLine

This method is *deprecated*. Replace this method with the *getListBoxData* method.

A.2.36. getMenuItemState

This method is *deprecated*. Use the *Menu* object methods directly.

A.2.37. getMouseCapture

This method is *deprecated*. Use the *Mouse* object methods directly.

A.2.38. getMultiList

This method is *deprecated*. Replace this method with the *getListBoxData* method.

A.2.39. getProgressBar

This method is *deprecated*. Replace this method with the *newProgressBar* method.

A.2.40. getRadioButton

This method is *deprecated*. Replace this method with the *getRadioButtonData* method.

A.2.41. getRadioControl

This method is *deprecated*. Replace this method with the *newRadioButton* method.

A.2.42. getScrollBar

This method is *deprecated*. Replace this method with the *newScrollBar* method.

A.2.43. getSliderControl

This method is *deprecated*. Replace this method with the *newTrackBar* method.

A.2.44. getStaticControl

This method is *deprecated*. Replace this method with the *newStatic* method.

A.2.45. getTabControl

This method is *deprecated*. Replace this method with the *newTab* method.

A.2.46. getTextSize

This method is *deprecated*. Replace this method with, preferably, the *getTextSizeDu* method, or the *getTextSizeDlg* method if absolutely necessary.

This method never worked correctly, the previous documentation for this method was incorrect and / or misleading. That has been fixed in the *getTextSizeDu*() and *getTextSizeDlg*() methods.

A.2.47. getTreeControl

This method is *deprecated*. Replace this method with the *newTreeView* method.

A.2.48. getValue

This method is *deprecated*. Replace this method with the *getControlData* method.

A.2.49. getWindowRect

This method is *deprecated*. Use the *windowRect* method instead.

A.2.50. grayMenuItem

This method is *deprecated*. Use the *Menu* object methods directly.

A.2.51. handleMessages

This method is *deprecated*. It sleeps for 10 milliseconds and returns.

A.2.52. hideItem

This method is *deprecated*. Replace this method with the *hideControl* method.

A.2.53. hideItemFast

This method is *deprecated*. Replace this method with the *hideControlFast* method.

A.2.54. isMouseButtonDown

This method is *deprecated*. Use the *Mouse* object methods directly.

A.2.55. itemText

This method is *deprecated*. Replace this method with the *setControlText* method. **Note** that this method has always been present in ooDialog, but was never previously documented.

A.2.56. moveItem

This method is *deprecated*. Use the *moveControl* method instead.

A.2.57. peekDialogMessage

This method is *deprecated*. It does nothing and always returns the empty string.

A.2.58. redrawItem

This method is *deprecated*. Replace this method with the *redrawControl* method.

A.2.59. releaseMouseCapture

This method is *deprecated*. Use the *Mouse* object methods directly.

A.2.60. resizeItem

This method is *deprecated*. Use the *resizeControl* method instead.

A.2.61. run

This method is *deprecated*. It does nothing except sit in a loop burning up CPU.

Although this method was documented previously, it is the author's opinion that it was intended for internal use only. Unfortunately, earlier versions of ooDialog documented a number of internal use only methods, which was a mistake. Internal use only methods should not be documented.

The earlier documentation did say: *run is a protected method. You cannot call this method directly.* In the author's opinion, this was poorly worded and what was meant to be said is: *run is an internal use only method. Do not call this method directly.*

A.2.62. scrollInButton

This method is *deprecated*. Replace this method with the *scrollInControl* method.

A.2.63. setItemFont

This method is *deprecated*. Replace this method with the *setControlFont* method.

A.2.64. setItemColor

This method is *deprecated*. Replace this method with the *setControlColor* method.

A.2.65. setItemSysColor

This method is *deprecated*. Replace this method with the *setControlSysColor* method.

A.2.66. setMenuItemRadio

This method is *deprecated*. Use the *Menu* object methods directly.

A.2.67. setStaticText

This method is *deprecated*. Replace this method with the *setControlText* method.

A.2.68. uncheckMenuItem

This method is *deprecated*. Use the *Menu* object methods directly.

A.2.69. writeToButton

This method is *deprecated*. Replace this method with the *writeToControl* method.

A.3. Deprecated Dialog Control Methods

A.3.1. Deprecated Button Methods

The following lists the *deprecated* methods of the *Button* class and how to replace those methods in ooDialog programs. The implementation of the following *bitmap* button methods was done in the *dialog* object. The methods are methods of the dialog, not the button.

A.3.1.1. changeBitmap

This method is *deprecated*. Replace this method with the *changeBitmapButton* method of the *dialog* object.

A.3.1.2. dimBitmap

This method is *deprecated*. Replace this method with the *dimBitmap* method of the *dialog* object.

A.3.1.3. displaceBitmap

This method is *deprecated*. Replace this method with the *displaceBitmap* method of the *dialog* object.

A.3.1.4. drawBitmap

This method is *deprecated*. Replace this method with the *drawBitmap* method of the *dialog* object.

A.3.1.5. getBitmapPosition

This method is *deprecated*. Replace this method with the *getBitmapPosition* method of the *dialog* object.

A.3.1.6. getBitmapSizeX

This method is *deprecated*. Replace this method with the *getBitmapSizeX* method of the *dialog* object.

A.3.1.7. getBitmapSizeY

This method is *deprecated*. Replace this method with the *getBitmapSizeY* method of the *dialog* object.

A.3.1.8. scroll

This method is *deprecated*. Replace this method with the *scrollButton* method of the *dialog* object.

A.3.1.9. scrollBitmapFromTo

This method is *deprecated*. Replace this method with the *scrollBitmapFromTo* method of the *dialog* object.

A.3.1.10. scrollText

This method is *deprecated*. Replace this method with the *scrollText* method of the *dialog* object.

A.3.1.11. setBitmapPosition

This method is *deprecated*. Replace this method with the *setBitmapPosition* method of the *dialog* object.

A.3.2. Deprecated ComboBox Methods

The following lists the *deprecated* methods of the *ComboBox* class and how to replace those methods in ooDialog programs.

A.3.2.1. editSelection

This method is *deprecated*. Use *setEditSelection*.

A.3.3. Deprecated ListView Methods

The following lists the *deprecated* methods of the *ListView* class and how to replace those methods in ooDialog programs.

A.3.3.1. removeImages

This method is deprecated. It is replaced by the functionally equivalent *setImageList* method. Do not use this method in new code. Try to migrate existing code to to the **`setImageList()`** method. This method may not exist in future versions of ooDialog.

A.3.3.2. removeSmallImages

This method is deprecated. It is replaced by the functionally equivalent *setImageList* method. Do not use this method in new code. Try to migrate existing code to to the **`setImageList()`** method. This method may not exist in future versions of ooDialog.

A.3.3.3. restoreEditClass

This method is deprecated. It is not needed and was for internal use only. Invoking the method is currently a NOP. This method may not exist in future versions of ooDialog.

A.3.3.4. setImages

This method is deprecated. It is replaced by the functionally equivalent *setImageList* method. Do not use this method in new code. Try to migrate existing code to to the **`setImageList()`** method. This method may not exist in future versions of ooDialog.

A.3.3.5. setSmallImages

This method is deprecated. It is replaced by the functionally equivalent *setImageList* method. Do not use this method in new code. Try to migrate existing code to to the **`setImageList()`** method. This method may not exist in future versions of ooDialog.

A.3.3.6. subclassEdit

This method is deprecated. It is not needed and was for internal use only. Invoking the method is currently a NOP. This method may not exist in future versions of ooDialog.

A.3.4. Deprecated ProgressBar Methods

A.3.4.1. getRange

This method is *deprecated*. Use*getFullRange*.

A.3.4.2. setRange

This method is *deprecated*. Use*setFullRange*.

A.3.5. Deprecated Radio Button Methods

A.3.5.1. isChecked

This method is *deprecated*. Replace this method with the *getCheckState* method for radio buttons, and the over-ridden *getCheckState* method for check boxes. This method was poorly named and in some circumstances produced inconsistent results.

A.3.5.2. indeterminate

This method is *deprecated*. Replace this method with the *setIndeterminate* method of the *CheckBox* class. Radio buttons can not be set to indeterminate. Using this method with a radio button sets it to the checked state, not the indeterminate state. In addition, this method was poorly named. One can check a button, but one does not indeterminate a check box.

A.3.6. Deprecated Tab Methods

The following lists the *deprecated* methods of the *Tab* class and how to replace those methods in ooDialog programs.

A.3.6.1. removeImages

This method is deprecated. It is replaced by the functionally equivalent *setImageList* method. Do not use this method in new code. Try to migrate existing code to to the **`setImageList()`** method. This method may not exist in future versions of ooDialog.

A.3.6.2. setImages

This method is deprecated. It is replaced by the functionally equivalent *setImageList* method. Do not use this method in new code. Try to migrate existing code to to the **`setImageList()`** method. This method may not exist in future versions of ooDialog.

A.3.7. Deprecated TreeView Methods

The following lists the *deprecated* methods of the *TreeView* class and how to replace those methods in ooDialog programs.

A.3.7.1. hitTest

This method is deprecated. It is replaced by the functionally equivalent *hitTestInfo* method. Do not use this method in new code. Try to migrate existing code to to the **`hitTestInfo()`** method. This method may not exist in future versions of ooDialog.

A.3.7.2. removeImages

This method is deprecated. It is replaced by the functionally equivalent *setImageList* method. Do not use this method in new code. Try to migrate existing code to to the **`setImageList()`** method. This method may not exist in future versions of ooDialog.

A.3.7.3. restoreEditClass

This method is deprecated. It is not needed and was for internal use only. Invoking the method is currently a NOP. This method may not exist in future versions of ooDialog.

A.3.7.4. setImages

This method is deprecated. It is replaced by the functionally equivalent *setImageList* method. Do not use this method in new code. Try to migrate existing code to to the **`setImageList()`** method. This method may not exist in future versions of ooDialog.

A.3.7.5. subclassEdit

This method is deprecated. It is not needed and was for internal use only. Invoking the method is currently a NOP. This method may not exist in future versions of ooDialog.

A.4. Deprecated DialogControl Methods

The following lists the *deprecated* methods of the *DialogControl* class and how to replace those methods in ooDialog programs.

A.4.1. captureMouse

This method is *deprecated*. Use the *Mouse* object methods directly.

A.4.2. getFocus

This method is *deprecated*. Use the *dialog* object's*getFocus* method instead.

A.4.3. getMouseCapture

This method is *deprecated*. Use the *Mouse* object methods directly.

A.4.4. getTextSize

This method is *deprecated*. Replace this method with the *getTextSizeDu* method. The *getTextSizeDlg* method could also be used, if absolutely necessary. However, that is not recommended.

This method never worked correctly, the previous documentation for this method was incorrect and / or misleading. That has been fixed in the *getTextSizeDlg*() method.

A.4.5. isMouseButtonDown

This method is *deprecated*. Use the *Mouse* object methods directly.

A.4.6. processMessage

This method is *deprecated*. Replace this method with the *sendMessage* method.

A.4.7. releaseMouseCapture

This method is *deprecated*. Use the *Mouse* object methods directly.

A.4.8. setFocus

This method is *deprecated*. Use the *dialog* object's*setFocus* method instead.

A.4.9. tabToNext

This method is *deprecated*. Use the *dialog* object's*tabToNext* method instead.

A.4.10. tabToPrevious

This method is *deprecated*. Use the *dialog* object's*tabToPrevious* method instead.

A.4.11. value

This method is *deprecated*. Replace this method with the *data* method.

A.4.12. value=

This method is *deprecated*. Replace this method with the *data =* method.

A.5. Deprecated Event Notification Methods

Earlier versions of ooDialog used a confusing array of method names with the word *connect* in them. After the release of ooRexx 4.0.0, an effort was made to unify the method names in ooDialog. The methods listed in this section are all *deprecated*. They all forward to the method that has taken their place. You can remove this level of indirection in your programs by replacing all of the deprecated methods.

A.5.1. connectButton

This method is *deprecated*. Replace this method using the *connectButtonEvent* method in this manner:

```
-- Change the following code:
self~connectButton(IDC_PB_REFRESH, onRefresh)

-- To this code:
```

```
self~connectListButtonEvent(IDC_PB_REFRESH, "CLICKED", onRefresh)
```

A.5.2. connectButtonNotify

This method is *deprecated*. Replace this method with the *connectButtonEvent* method.

A.5.3. connectComboBoxNotify

This method is *deprecated*. Replace this method with the *connectComboBoxEvent* method.

A.5.4. connectCommonNotify

This method is *deprecated*. Replace this method with the *connectNotifyEvent* method.

A.5.5. connectControl

This method is *deprecated*. Replace this method with the *connectCommandEvents* method.

A.5.6. connectEditNotify

This method is *deprecated*. Replace this method with the *connectEditEvent* method.

A.5.7. connectList

This method is *deprecated*. Replace this method by using the *connectListViewEvent* method in this manner:

```
-- Change the following code:
self~connectList(IDC_LISTBOX, onList)

-- To this code:
self~connectListBoxEvent(IDC_LISTBOX, "SELCHANGE", onList)
```

A.5.8. connectListBoxNotify

This method is *deprecated*. Replace this method with the *connectListBoxEvent* method.

A.5.9. connectListLeftDoubleClick

This method is *deprecated*. Replace this method by using the *connectListViewEvent* method in this manner:

```
-- Change the following code:
self~connectListLeftDoubleClick(IDC_LISTBOX, onDoubleClick)

-- To this code:
self~connectListBoxEvent(IDC_LISTBOX, "DLBCLK", onDoubleClick)
```

A.5.10. connectListNotify

This method is *deprecated*. Replace this method with the *connectListViewEvent* method.

A.5.11. connectListViewNotify

This method is *deprecated*. Replace this method with the *connectListViewEvent* method.

A.5.12. connectMenuItem

This method is *deprecated*. Use the *Menu* object methods directly.

A.5.13. connectMouseCapture

This method is *deprecated*. Use the *Mouse* object methods directly.

A.5.14. connectScrollBar

This method is *deprecated*. Replace this method with the *connectEachSBEvent* method.

A.5.15. connectScrollBarNotify

This method is *deprecated*. Replace this method with the *connectScrollBarEvent* method.

A.5.16. connectSliderNotify

This method is *deprecated*. Replace this method with the *connectTrackBarEvent* method.

A.5.17. connectStaticNotify

This method is *deprecated*. Replace this method with the *connectStaticEvent* method.

A.5.18. connectTabNotify

This method is *deprecated*. Replace this method with the *connectTabEvent* method.

A.5.19. connectTreeNotify

This method is *deprecated*. Replace this method with the *connectTreeViewEvent* method.

A.6. Deprecated External Functions

Do not directly use any of the ooDialog external functions.

The ooDialog documentation prior to ooDialog 4.0.0 documented ten external functions as *callable functions that can be used in your Object Rexx programs*. It then mentioned in passing that the Public *Routines* were a more convenient way to use these functions. It was noted that the external routines were registered automatically when the first dialog was initialized, but if the programmer wanted to use the external routines when no dialog was created, he could register the individual functions using **RxFuncAdd**.

The 3.2.0 documentation expounded on that a little and also mentioned that there were other external functions, noting that it would be difficult for ooDialog programmers to use these other functions unless they were experienced C programmers. In hindsight that was a mistake. Each of the external functions has a matching public routine. No mention of the external functions should have been made. Instead, only the public routines should have been documented.

The reason for this is simple, the external functions have been removed. Prior to 4.0.0, ooDialog was implemented using the original external native API. This API was useful, but somewhat limiting. Particularly when used with the object-oriented paradigm. ooRexx 4.0.0 introduces a new native API that is much more robust and flexible. Using this new API makes writing external packages like ooDialog much easier.

ooDialog has been completely converted to use this new API. ***All** of the external functions have been removed.* If your programs directly uses any of the **undocumented** external functions, they will not work with the 4.2.0 release of ooDialog.

The documented external functions will be available as **deprecated** functions, with the possibility of their removal in the future. However, they will simply map to the corresponding public routine. The programmer can remove this extra level of indirection by calling the public routine directly. It would be nice if ooDialog did not have to drag along this baggage forever into the future.

It is no longer necessary to use RxFuncAdd(). Doing so is a nop. Likewise, registering InstMMFuncs() and calling it does nothing. The ooDialog programmer should remove any references to RxFuncAdd (when used to register any ooDialog external function,) and InstMMFuncs in their code as the opportunity arises.

A.6.1. errorMessage

This function is *deprecated*. Use the public routine *ErrorDialog*.

A.6.2. getFileNameWindow

This function is deprecated. See this *explanation*. Use the public routine *FileNameDialog*. Do not use this function in new code. Try to migrate existing code to FileNameDialog. This function may not exist in future versions of ooDialog.

A.6.3. getScreenSize

This function is deprecated. See this *explanation*. Use the public routine *ScreenSize*. Do not use this method in new code. Try to migrate existing code to screenSize. This function may not exist in future versions of ooDialog.

A.6.4. getSysMetrics

This function is deprecated. It is replaced by the functionally equivalent *getSystemMetrics* method of the .DlgUtil class. Do not use this function in new code. Try to migrate existing code to to the **`.DlgUtil~getSystemMetrics()`** method. This function may not exist in future versions of ooDialog.

A.6.5. infoMessage

This function is *deprecated*. Use the public routine *InfoDialog*.

A.6.6. playSoundFile

This function is deprecated. See this *explanation*. Use the public routine *Play*. Do not use this function in new code. Try to migrate existing code to Play. This function may not exist in future versions of ooDialog.

A.6.7. playSoundFileInLoop

This function is deprecated. See this *explanation*. Use the public routine *Play*. Do not use this function in new code. Try to migrate existing code to *play*(). This function may not exist in future versions of ooDialog.

A.6.8. sleepMS

This function is *deprecated*. Use the public routine *MSSleep*.

A.6.9. stopSoundFile

This function is deprecated. See this *explanation*. Use the public routine *Play*. Do not use this function in new code. Try to migrate existing code to *play*(). This function may not exist in future versions of ooDialog.

A.6.10. systemMetrics Routine

This routine is deprecated. It is replaced by the functionally equivalent *getSystemMetrics* method of the .DlgUtil class. Do not use this routine in new code. Try to migrate existing code to to the **`.DlgUtil~getSystemMetrics()`** method. This routine may not exist in future versions of ooDialog.

A.6.11. yesNoMessage

This function is *deprecated*. Use the public routine *AskDialog*.

A.7. Deprecated UserDialog Methods

Deprecated **`UserDialog`** methods and their replacement methods are listed in this section.

A.7.1. addBitmapButton

This method is *deprecated*. Replace this method with the *createBitmapButton* method.

A.7.2. addBlackFrame

This method is *deprecated*. Replace this method with the *createBlackFrame* method.

A.7.3. addBlackRect

This method is *deprecated*. Replace this method with the *createBlackRect* method.

A.7.4. addButton

This method is *deprecated*. Replace this method with the *createPushButton* method.

A.7.5. addButtonGroup

This method is *deprecated*. Replace this method with the *createPushButtonGroup* method.

A.7.6. addCheckBox

This method is *deprecated*. Replace this method with the *createCheckBox* method.

A.7.7. addCheckBoxStem

This method is *deprecated*. Replace this method with the *createCheckBoxStem* method.

A.7.8. addCheckGroup

This method is *deprecated*. Replace this method with the *createCheckBoxGroup* method.

A.7.9. addComboBox

This method is *deprecated*. Replace this method with the *createComboBox* method.

A.7.10. addComboInput

This method is *deprecated*. Replace this method with the *createComboBoxInput* method.

A.7.11. addEntryLine

This method is *deprecated*. Replace this method with the *createEdit* method.

A.7.12. addEtchedFrame

This method is *deprecated*. Replace this method with the *createEtchedFrame* method.

A.7.13. addEtchedHorizontal

This method is *deprecated*. Replace this method with the *createEtchedHorizontal* method.

A.7.14. addEtchedVertical

This method is *deprecated*. Replace this method with the *createEtchedVertical* method.

A.7.15. addGrayFrame

This method is *deprecated*. Replace this method with the *createGrayFrame* method.

A.7.16. addGrayRect

This method is *deprecated*. Replace this method with the *createGrayRect* method.

A.7.17. addGroupBox

This method is *deprecated*. Replace this method with the *createGroupbox* method.

A.7.18. addIcon

This method is *deprecated*. Replace this method with the *addIconResource* method.

A.7.19. addImage

This method is *deprecated*. Replace this method with the *createStaticImage* method.

A.7.20. addInput

This method is *deprecated*. Replace this method with the *createEditInput* method.

A.7.21. addInputGroup

This method is *deprecated*. Replace this method with the *createEditInputGroup* method.

A.7.22. addInputStem

This method is *deprecated*. Replace this method with the *createEditInputStem* method.

A.7.23. addListBox

This method is *deprecated*. Replace this method with the *createListBox* method.

A.7.24. addListControl

This method is *deprecated*. Replace this method with the *createListView* method.

A.7.25. addMenuItem

This method is *deprecated*. Use the *Menu* object methods directly.

A.7.26. addMenuSeparator

This method is *deprecated*. Use the *Menu* object methods directly.

A.7.27. addOkCancelLeftBottom

This method is *deprecated*. Replace this method with the *createOkCancelLeftBottom* method.

A.7.28. addOkCancelLeftTop

This method is *deprecated*. Replace this method with the *createOkCancelLeftTop* method.

A.7.29. addOkCancelRightBottom

This method is *deprecated*. Replace this method with the *createOkCancelRightBottom* method.

A.7.30. addOkCancelRightTop

This method is *deprecated*. Replace this method with the *createOkCancelRightTop* method.

A.7.31. addPasswordLine

This method is *deprecated*. Replace this method with the *createPasswordEdit* method.

A.7.32. addPopupMenu

This method is *deprecated*. Use the *Menu* object methods directly.

A.7.33. addProgressBar

This method is *deprecated*. Replace this method with the *createProgressBar* method.

A.7.34. addRadioButton

This method is *deprecated*. Replace this method with the *createRadioButton* method.

A.7.35. addRadioGroup

This method is *deprecated*. Replace this method with the *createRadioButtonGroup* method.

A.7.36. addRadioStem

This method is *deprecated*. Replace this method with the *createRadioButtonStem* method.

A.7.37. addScrollBar

This method is *deprecated*. Replace this method with the *createScrollBar* method.

A.7.38. addSliderControl

This method is *deprecated*. Replace this method with the *createTrackbar* method.

A.7.39. addStatic

This method is *deprecated*. Replace this method with the *createStatic* method.

A.7.40. addTabControl

This method is *deprecated*. Replace this method with the *createTab* method.

A.7.41. addText

This method is *deprecated*. Replace this method with the *createStaticText* method.

A.7.42. addTreeControl

This method is *deprecated*. Replace this method with the *createTreeView* method.

A.7.43. addWhiteFrame

This method is *deprecated*. Replace this method with the *createWhiteFrame* method.

A.7.44. addWhiteRect

This method is *deprecated*. Replace this method with the *createWhiteRect* method.

A.7.45. createMenu

This method is *deprecated*. Use the *Menu* object methods directly.

A.7.46. loadMenu

This method is *deprecated*. Use the *Menu* object methods directly.

A.7.47. setMenu

This method is *deprecated*. Use the *Menu* object methods directly.

A.8. Deprecated WindowBase Methods

The *WindowBase* class is inherited by both the dialog *dialog* object and the dialog *control* object. Therefore, depending on the point of view of the reader, these deprecated methods may be thought of as either deprecated dialog methods or deprecated dialog control methods.

The following lists the *deprecated* methods of the *WindowBase* class and how to replace those methods in ooDialog programs.

A.8.1. assignWindow

This method is deprecated and always returns 0, *the assignment failed*. To work with a dialog or dialog control object, construct the proper ooDialog object. This ensures that the object has the correct methods. To control a window not owned by your program use the *find*() method of the *WindowsManager* class. The *WindowsManager*is part of the *WinSystm.cls* package. In certain cases it may be desirable to invoke a method common to all windows when only the window *handle* is known. For these cases use a *Window* object.

A.8.2. getSystemMetrics

This method is *deprecated*. Replace this method the functionally equivalent *getSystemMetrics* method of the *.DlgUtil*class.

A.9. Deprecated WindowExtensions Methods

The *WindowsExtensions* class is inherited by both the *dialog* object and the dialog *control* object. Therefore, depending on the point of view of the reader, these deprecated methods may be thought of as either deprecated dialog methods or deprecated dialog control methods.

The following lists the *deprecated* methods of the *WindowExtensions* class and how to replace those methods in ooDialog programs.

A.9.1. absRect2LogRect

This method is *deprecated* because it is incorrectly implemented and *inaccurate*. Replace this method with the *pixel2dlgUnit* method.

A.9.2. cursor_appStarting

This method is *deprecated*. Use the *Mouse* object methods directly.

A.9.3. cursor_arrow

This method is *deprecated*. Use the *Mouse* object methods directly.

A.9.4. cursor_cross

This method is *deprecated*. Use the *Mouse* object methods directly.

A.9.5. cursor_no

This method is *deprecated*. Use the *Mouse* object methods directly.

A.9.6. cursor_wait

This method is *deprecated*. Use the *Mouse* object methods directly.

A.9.7. cursorPos

This method is *deprecated*. Use the *Mouse* object methods directly.

A.9.8. logRect2AbsRect

This method is *deprecated* because it is incorrectly implemented and *inaccurate*. Replace this method with the *dlgUnit2pixel* method.

A.9.9. restoreCursorShape

This method is *deprecated*. Use the *Mouse* object methods directly.

A.9.10. setCursorPos

This method is *deprecated*. Use the *Mouse* object methods directly.

Appendix B. Notices

Any reference to a non-open source product, program, or service is not intended to state or imply that only non-open source product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Rexx Language Association (RexxLA) intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-open source product, program, or service.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-open source products was obtained from the suppliers of those products, their published announcements or other publicly available sources. RexxLA has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-RexxLA packages. Questions on the capabilities of non-RexxLA packages should be addressed to the suppliers of those products.

All statements regarding RexxLA's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

B.1. Trademarks

Open Object Rexx™ and ooRexx™ are trademarks of the Rexx Language Association.

The following terms are trademarks of the IBM Corporation in the United States, other countries, or both:

1-2-3
AIX
IBM
Lotus
OS/2
S/390
VisualAge

AMD is a trademark of Advance Micro Devices, Inc.

Intel, Intel Inside (logos), MMX and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the Unites States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

B.2. Source Code For This Document

The source code for this document is available under the terms of the Common Public License v1.0 which accompanies this distribution and is available in the appendix *Appendix C, Common Public License Version 1.0*. The source code is available at *https://sourceforge.net/p/oorexx/code-0/HEAD/tree/docs/*.

The source code for this document is maintained in DocBook SGML/XML format.

The railroad diagrams were generated with the help of "Railroad Diagram Generator" located at *http://bottlecaps.de/rr/ui*. Special thanks to Gunther Rademacher for creating and maintaining this tool.

Appendix C. Common Public License Version 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMMON PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

C.1. Definitions

"Contribution" means:

1. in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
2. in the case of each subsequent Contributor:
   a. changes to the Program, and
   b. additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents " mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

C.2. Grant of Rights

1. Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.
2. Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.
3. Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement

of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

4. Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

C.3. Requirements

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

1. it complies with the terms and conditions of this Agreement; and
2. its license agreement:
    a. effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
    b. effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
    c. states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
    d. states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

1. it must be made available under this Agreement; and
2. a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

C.4. Commercial Distribution

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified

Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

C.5. No Warranty

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

C.6. Disclaimer of Liability

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

C.7. General

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, if Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable.

However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. IBM is the initial Agreement Steward. IBM may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

Appendix D. Revision History

Revision 0-0 **Tue Aug 7 2012** **David Ashley**

Initial creation of book by publican

Index

Symbols

+, 1624
-, 1624
.application, 1581
.constDir, 1592
.systemErrorCode, 1651

A

absoluteWidth, 1508
activate, 1265
add, 1316
    ComboBox class, 884
    ImageList class, 1476
    ListBox class, 963
    ListView class, 983
addAutoStartMethod, 233
addBitmap, 1238
addButtons, 1238
addCheckButton, 1701
addComboBox, 1702
addComboEntry, 151
addControlItem, 1702
addDirectory
    ComboBox class, 885
    ListBox class, 963
addEditBox, 1703
addExtendedStyle, 984
addFullRow, 986
addFullSeq, 1222
addIcon
    ImageList class, 1477
addIconResource, 693
addImages
    ImageList class, 1477
addItem, 1433
AddLine
    InputBox class, 1501
    PasswordBox class, 1505
addListEntry, 190
addMasked
    ImageList class, 1479
addMenu, 1704
addNewAttribute, 95
addNewMethod, 96
addPage, 590
addPlace, 1727
addPopup, 1435
addPushButton, 1705
addRadioButtonList, 1706
addRow, 987
addRowFromArray, 988
addSeparator
    CommonDialogCustomizations class, 1706
    UserMenuBar class, 1437
addSequence, 1223
addString, 1239
addStyle
    Edit class, 929
    ListView class, 989
    MonthCalendar class, 1112
addSubItem, 1081
addText, 1707
addToConstDir, 1584
addTool, 1266
addToolEx, 1268
addToolRect, 1268
addUserMsg, 301
    event handlers
        other messages, 306
        WM_COMMAND messages, 307
        WM_NOTIFY messages, 309
adjustRect, 1270
adjustToRectangle, 1223
advise, 1728
Alerter class, 1520
    interrupt, 1521
    interruptible, 1522
    isCanceled, 1521
    new, 1520
alignLeft, 990
alignTop, 990
and, 1594
AnimatedButton class, 856
appearance methods
    dialog object, 125
appIcon, 585
ApplicationManager class, 1581
    addToConstDir, 1584
    autoDetection, 1585
    defaultFont, 1585
    defaultIcon, 1586
    new, 1582
    parseIncludeFile, 489, 489, 490
    requiredOS, 1587
    setDefaults, 1588
    srcDir, 1582
    useGlobalConstDir, 1590
apply
    PropertySheetDialog class, 591
    PropertySheetPage class, 621
appStarting, 1549
arrange, 990
arrow, 1551
AskDialog, 1528
assignBitmapID, 1260

assignFocus, 830
assignTo, 1443
attachTo, 1423
autoDetect, 73
autoDetection, 161, 1585
autoSize, 1240

B

b, 650
backgroundBitmap, 148
backgroundColor, 125
  ProgressBar class, 1152
backgroundSysColor, 125
banner, 1681
barColor
  ProgressBar class, 1153
Base classes, 212
Base Mixin classes, 212
begin, 1515
behavior methods
  dialog object, 125
BinaryMenuBar class, 1427
bitmap methods, 148
bitmapBack, 1182
bitmapID, 1258
bkColor, 990
bkColor=, 991
bottom
  Rect class, 1630
BrowseForFolder
  Special Folder Names, 1675
BrowseForFolder class, 1679
  banner, 1681
  dlgTitle, 1681
  getDisplayName, 1690
  getFolder, 1691
  getItemIDList, 1692
  hint, 1682
  initCOM, 1694
  initialThread, 1683
  new, 1680
  options, 1685
  owner, 1687
  releaseCOM, 1695
  releaseItemIDList, 1697
  root, 1688
  Shell / COM considerations, 1672
  startDir, 1689
  usePathForHint, 1689
Button class, 867
  click, 869
  events
    CLICKED, 315
    DBLCLK, 315
    DISABLE, 315
    DROPDOWN, 314
    GOTFOCUS, 315
    HILITE, 315
    HOTITEM, 316
    LOSTFOCUS, 315
    PAINT, 315
    UNHILITE, 315
  getIdealSize, 869
  getImage, 870
  getImageList, 870
  getTextMargin, 871
  push, 872
  setImage, 873
  setImageList, 873
  setTextMargin, 876
  state, 876
  state=, 877
  style=, 878
Button Controls, 856
buttonCount, 1240

C

calcDisplayRect, 1224
calcWindowRect, 1225
CanCancel, 1509
cancel, 122
cancelToClose, 591
canUndo, 930
caption, 586
captionText, 1509
capture, 1551
center, 126
changeBitmap, 1241
changeBitmapButton, 233
changeComboEntry, 151
changed, 591
changeListEntry, 190
check
  CommonDialogCustomizations class, 1708
  ListView class, 991
  menu object, 1378
  RadioButton class, 865
checkAll, 992
CheckBox class, 859
  getCheckState, 861
  isIndeterminate, 861
  setIndeterminate, 862
checkButton, 1241
checked, 866
checkInGroup, 865
CheckList
  class, 1499
  function, 1528

checkRadio, 1379
chevronLocation, 1182
chevronState, 1183
child, 1183
Child, 1318
childWindowFromPoint, 497
class methods
  dialog object, 69
clear, 498
clearClientData, 1729
clearControlRect, 235
clearRect
  dialog control object, 831
  dialog object, 236
clearSelRange, 1304
clearText, 930
clearTicks, 1305
clearWindowRect, 238
click, 869
client2screen, 499
clientRect, 499
clientToScreen, 500
clipCursor, 1552
close, 1729
closeDropDown, 886
closeMonthCal, 911
clrBack, 1184
clrFore, 1184
clrText, 1069, 1350
clrTextBk, 1069, 1350
cmdID, 1258
cMouseButtons, 1639
coding event handler examples, 295
coding event handlers, 293
Collapse, 1319
CollapseAndReset, 1319
colorRef, 1461
columnFormats, 1091
columnInfo, 992
columns, 1092
columnWidth, 993
columnWidth=, 964
columnWidthPX, 993
columnWidthPx=, 964
combo box methods, 151
comboAddDirectory, 152
ComboBox class, 882
  add, 884
  addDirectory, 885
  closeDropDown, 886
  delete, 886
  deleteAll, 887
  events
    CHANGE, 319
    CLOSEUP, 319
    DBLCLK, 319
    DROPDOWN, 319
    ERRSPACE, 319
    GOTFOCUS, 319
    LOSTFOCUS, 319
    SELCHANGE, 319
    SELENDCANCEL, 319
    SELENDOK, 319
    UPDATE, 319
  find, 887
  getCombBoxInfo, 889
  getCue, 888
  getEditControl, 890
  getFirstVisible, 891
  getHorizontalExtent, 891
  getItemHeight, 892
  getMinVisible, 892
  getText, 893
  insert, 893
  isDropDown, 894
  isDropDownList, 894
  isDropDownOpen, 895
  isGrandchild, 895
  isSimple, 896
  items, 897
  modify, 897
  openDropDown, 898
  removeFullColor, 898
  select, 899
  selected, 899
  selectedIndex, 900
  selectIndex, 900
  setCue, 900
  setEditSelection, 901
  setFullColor, 902
  setHorizontalExtent, 903
  setItemHeight, 904
  setMinVisible, 905
comboDrop, 153
ComConstants mixin class, 1699
comCtl32Version, 1595
commandToIndex, 1242
Common Control Styles, 702
Common Input Dialogs and Routines, 1497
Common Public License, 1798
Common willReply argument, 299
CommonDialogCustomization class
  addMenu, 1704
CommonDialogCustomizations class, 1699
  addCheckButton, 1701
  addComboBox, 1702
  addControlItem, 1702
  addEditBox, 1703

addPushButton, 1705
addRadioButtonList, 1706
addSeparator, 1706
addText, 1707
check, 1708
enableOpenDropDown, 1708
endVisualGroup, 1709
getCheckButtonState, 1710
getControlItemState, 1710
getControlState, 1711
getEditBoxText, 1712
getSelectedControlItem, 1713
isChecked, 1714
makeProminent, 1714
removeAllControlItems, 1715
removeControlItem, 1716
setCheckButtonState, 1716
setControlItemState, 1717
setControlItemText, 1718
setControlLabel, 1718
setControlState, 1719
setEditBoxText, 1720
setSelectedControlItem, 1721
startVisualGroup, 1721
uncheck, 1722
CommonDialogEvents class, 1751
constant methods, 1752
new, 1752
onButtonClicked, 1753
onCheckButtonToggled, 1754
onControlActivating, 1754
onFileOk, 1755
onFolderChange, 1756
onFolderChanging, 1757
onHelp, 1758
onItemSelected, 1758
onOverwrite, 1760
onSelectionChange, 1761
onShareViolation, 1761
onTypeChange, 1762
CommonItemDialog class, 1723
addPlace, 1727
advise, 1728
clearClientData, 1729
close, 1729
getCurrentSelection, 1731
getFileName, 1732
getFileTypeIndex, 1733
getFolder, 1733
getResult, 1734
initCOM, 1735
isReleased, 1737
new, 1724
options, 1725
release, 1737
releaseCOM, 1739
setCancelButtonLabel, 1740
setClientGuid, 1741
setDefaultExtension, 1742
setDefaultFolder, 1743
setFileName, 1744
setFileNameLabel, 1744
setFileTypeIndex, 1745
setFileTypes, 1746
setFilter, 1747
setFolder, 1747
setOkButtonLabel, 1748
setTitle, 1749
show, 1750
unadvise, 1750
complete, 1438, 1515
connect data attribute methods, 161
connect event methods
dialog object, 116
connectActivate, 310
connectAllCommandEvents
menu object, 1380
SystemMenu class, 1454
connectAllSBEvents, 404
connectButtonEvent, 312
connectCharEvent, 832
connectCheckBox, 162
connectComboBox, 162
connectComboBoxEvent, 317
connectCommandEvent
event handler, 322
menu object
class method, 1376
instance method, 1381
SystemMenu class, 1455
connectCommandEvents
EventNotification class, 320
connectContextMenu
PopupMenu class
class method, 1441
instance method, 1444
connectDateTimePicker, 163
connectDateTimePickerEvent, 324
connectDraw, 337
connectEachSBEvent, 404
connectEdit, 164
connectEditEvent, 342
connectEvent, 1553
connectFKeyPress
dialog control object, 835
dialog object, 348
connectHelp, 345
connecting data attributes, 156

connecting event handlers, 293
connecting events, 116
Connecting Key Press Events, 348
Connecting Scroll Bar Events, 404
connectKeyPress
  dialog control object, 837
  dialog object, 350
connectListBox, 165
connectListBoxEvent, 358
connectListView, 170
connectListViewEvent, 360
connectMenuEvent, 1382
connectMonthCalendar, 171
connectMonthCalendarEvent, 381
connectMove, 386
connectNotifyEvent, 388
  events
    CLICK, 390
    DBLCLK, 390
    ENTER, 390
    GOTFOCUS, 390
    LOSTFOCUS, 390
    OUTOFMEMORY, 390
    RCLICK, 390
    RDBLCLK, 390
connectPosChanged, 391
connectRadioButton, 172
connectReBarEvent, 392
connectResize, 397
connectResizing, 399
connectScrollBarEvent, 406
connectSizeMoveEnded, 410
connectSomeCommandEvents
  menu object, 1388
  SystemMenu class, 1457
connectStaticEvent, 412
connectStatusBarEvent, 415
connectTab, 173
connectTabEvent, 419
connectToolBarEvent, 423
connectToolTipEvent, 428
connectTrackBar, 174
connectTrackBarEvent, 433
connectTreeView, 175
connectTreeViewEvent, 436
connectUpDown, 175
connectUpDownEvent, 457
constant methods
  CommonDialogEvents class, 1752
  CustomDraw class, 221
  dialog object, 68
  ListView class, 981
  Mouse class, 1543
  ProgressDialog class, 1507
  SystemMenu class, 1453
  TrackBar class, 1303
constDir, 488
Control dialogs, 571
Control Dialogs, 571
  ControlDialog class, 571
  RcControlDialog class, 574
  ResControlDialog class, 577
  UserControlDialog class, 578
controlBottom, 669
ControlDialog Class, 571
ControlDialog class
  endExecution, 572
  execute, 572
  initUpdateListView, 573
  wasActivated, 574
controlLeft, 670
controlRight, 671
controlSizing, 672
controlTop, 673
copy
  PointClass, 1625
  RectClass, 1631
copyText, 931
correctionFactor, 658
countTicks, 1305
CPL, 1798
create, 694
  ImageList class, 1474
create button controls, 703
create combo box controls, 735
create edit controls, 742
create static control methods, 769
create... methods, 699
createBitmapButton, 703
createBlackFrame, 769
createBlackRect, 771
createBrush
  DialogExtensions, 238
  WindowExtensions, 541
createCenter, 697
createCheckBox, 706
createCheckBoxGroup, 709
createCheckBoxStem, 712
createComboBox, 736
createComboBoxInput, 738
createDateTimePicker, 740
createEdit, 742
createEditInput, 745
createEditInputGroup, 748
createEditInputStem, 751
createEtchedFrame, 772
createEtchedHorizontal, 773
createEtchedVertical, 774

createFont, 542
createFontEx, 543
createGrayFrame, 775
createGrayRect, 776
createGroupbox, 715
createListBox, 757
createListView, 760
createMonthCalendar, 762
createOkCancelLeftBottom, 716
createOkCancelLeftTop, 717
createOkCancelRightBottom, 717
createOkCancelRightTop, 718
createPasswordEdit, 755
createPen, 550
createProgressBar, 764
createPushButton, 718
createPushButtonGroup, 721
createPushButtonStem, 723
createRadioButton, 727
createRadioButtonGroup, 730
createRadioButtonStem, 732
createReBar, 765
createReBarWindow, 213
createScrollBar, 768
createStatic, 777
createStaticFrame, 781
createStaticImage, 783
createStaticText, 785
createStatusBar, 791
createStatusBarWindow, 215, 217
createTab, 793
createToolBar, 795
createToolTip, 102
createTrackbar, 797
createTreeView, 799
createUpDown, 802
createWhiteFrame, 789
createWhiteRect, 790
CreateWindows class, 213
  createReBarWindow, 213
  createStatusBarWindow, 215, 217
cross, 1564
custom draw
  in ooDialog, 220
  list-view control, 1066
  overview, 218
  simple custom draw, 221
  tree-view control, 1348
customDraw, 223
CustomDraw class, 218
  constants, 221
  customDraw, 223
  customDrawControl, 224
  rgb, 225
customDrawControl, 224
customize, 1242
cutText, 931
cx, 613, 651, 1185
cxCursor, 1639
cxDrag, 1639
cxFixedFrame, 1640
cxHeader, 1185
cxIcon, 1640
cxIdeal, 1186
cxMinChild, 1186
cxScreen, 1640
cxSize, 1641
cxSmIcon, 1641
cxVscroll, 1641
cy, 614, 651
cyCaption, 1641
cyChild, 1187
cyCursor, 1642
cyDrag, 1642
cyFixedFrame, 1642
cyHscroll, 1643
cyIcon, 1643
cyIntegral, 1187
cyMaxChild, 1187
cyMenu, 1643
cyMinChild, 1188
cyScreen, 1643
cySize, 1644
cySmIcon, 1644

D

data attribute names, 157
data attribute problems, 161
data attributes, 156
data method, 841
data=, 842
date, 1111
DateTimePicker
  getMonthCalFont, 916
  setMonthCalFont, 922
DateTimePicker class, 907
  closeMonthCal, 911
  getDateTime, 911
  getIdealSize, 912
  getInfo, 912
  getMonthCal, 914
  getMonthCalColor, 915
  getMonthCalStyle, 917
  getRange, 918
  setDateTime, 920
  setFormat, 920
  setMonthCalColor, 921
  setMonthCalStyle, 922

setRange, 925
DateTimePicker Event
CLOSEUP, 326
DATETIMECHANGE, 328
DROPDOWN, 329
FORMAT, 330
FORMATQUERY, 332
KEYDOWN, 333
KILLFOCUS, 334
SETFOCUS, 335
USERSTRING, 335
DayState class, 1140
new, 1140
value, 1141
DayStates class, 1142
endMonth, 1145
getDayState, 1146
getDayStateBuffer, 1147
makeDayStateBuffer, 1142
new, 1144
putMonth, 1148
putYear, 1149
quickDayStateBuffer, 1144
startMonth, 1146
decr, 1626
defaultBottom, 674
defaultFont, 1585
defaultIcon, 1586
defaultLeft, 675
defaultRight, 676
defaultSizing, 677
defaultTop, 678
Define
TimedMessage, 1527
defineDialog, 804
InputBox class, 1501
defineSizing, 679
defListDragHandler, 459
defTreeDragHandler, 460
delete
ComboBox class, 886
ListBox class, 964
ListView class, 994
Tab class, 1225
TreeView class, 1320
deleteAll
ComboBox class, 887
ListBox class, 965
ListView class, 994
Tab class, 1226
TreeView class, 1320
deleteBand, 1160
deleteColumn, 994
deleteComboEntry, 153
deleteFont, 551
deleteListEntry, 190
deleteObject, 551
deletePopup, 1389
deltaPosReply, 1357
delTool, 1271
Deprecated, 1775
AdvancedControls Class, 1775
button methods, 1782
CategoryDialog Class, 1776
classes, 1775
combo box methods, 1783
dialog control methods, 1782
dialog control object methods, 1785
dialog object methods, 1776
EventNotification methods, 1786
external functions, 1788
list-view methods, 1783
MessageExtensions Class, 1776
MessageExtensions methods, 1786
PlainUserDialog Class, 1776
progress bar methods, 1784
PropertySheet Class, 1776
radio button methods, 1784
tab methods, 1784
tree-view methods, 1784
UserDialog methods, 1790
WindowBase methods, 1793
WindowExtensions methods, 1794
deselect, 995
deselectAll, 996
deSelectIndex, 965
deSelectRange, 965
destroy
menu object, 1390
detach, 1425
determinePosition, 1194
determineSBPosition, 239
Dialog Control class
useUnicode, 853
dialog control object, 809, 809
assignFocus, 830
attributes, 813
basic window methods, 816
clear, 498
clearRect, 831
client2screen, 499
clientRect, 499
clientToScreen, 500
connectCharEvent, 832
connectFKeyPress, 835
connectKeyPress, 837
createBrush, 541
createFont, 542

createFontEx, 543
createPen, 550
data method, 841
data=, 842
deleteFont, 551
deleteObject, 551
disable, 500
disconnectKeyPress, 843
display, 501
draw, 502
drawAngleArc, 551
drawArc, 552
drawLine, 553
drawPie, 553
drawPixel, 553
enable, 503
extended window methods, 824
factorX, 493
factorY, 493
fillDrawing, 554
fillRect, 555
fontColor, 556
fontToDC, 556
foregroundWindow, 503
freeDC, 557
getArcDirection, 557
getClientRect, 504
getDC, 557
getExStyleRaw, 504
getFont, 558
getID, 505
getPixel, 558
getPos, 505
getRealPos, 506
getRealSize, 506
getRect, 507
getSize, 507
getStyleRaw, 508
getSysBrush, 559
getText, 509
getTextAlign, 559
getTextExtent, 560
getTextSizeDlg, 844
getTextSizePx, 509, 510
group, 845
hasKeyPressConnection, 846
hDlg, 814
hide, 511
hideFast, 512
hScrollPos, 561
hwnd, 493
id, 814
initCode, 494
isEnabled, 512
isVisible, 513
loadBitmap, 561
move, 514
moveTo, 515
moveWindow, 517
objectToDC, 562
oDlg, 815
opaqueText, 562
pixelCX, 494
pixelCY, 495
rectangle, 563
redrawClient, 520
redrawRect, 847
removeBitmap, 563
resize, 520
resizeTo, 522
screen2client, 523
screenToClient, 524
scroll, 564
sendMessage, 524, 525
setArcDirection, 564
setColor, 847
setFont, 565
setHScrollPos, 565
setRect, 526
setStyleRaw, 528
setSysColor, 850
setText, 529
setTextAlign, 566
setTitle, 530
setVScrollPos, 567
setWindowPos, 530
setWindowTheme, 850
show, 533
showFast, 534
sizeWindow, 534
tabStop, 851
textSize, 852
title, 537
title=, 537
transparentText, 567
update, 537
updateWindow, 538
useUnicode, 853
useVersion, 853
usingUnicode, 854
usingVersion, 854
vScrollPos, 567
windowRect, 539
write, 567
writeDirect, 570
dialog control object Event
CHAR, 833
dialog object

addNewAttribute, 95
addNewMethod, 96
addUserMsg, 301
autoDetect, 73
autoDetection, 161
backgroundColor, 125
backgroundSysColor, 125
basic window methods, 80
cancel, 122
changeBitmapButton, 233
clear, 498
clearControlRect, 235
clearRect, 236
client2screen, 499
clientRect, 499
clientToScreen, 500
coding event handler examples, 295
coding event handlers, 293
connect data attribute methods, 161
connect event methods, 116
connectActivate, 310
connectAllSBEvents, 404
connectButtonEvent, 312
connectCheckBox, 162
connectComboBox, 162
connectComboBoxEvent, 317
connectCommandEvents, 320
connectDateTimePicker, 163
connectDateTimePickerEvent, 324
connectDraw, 337
connectEdit, 164
connectHelp, 345
connecting event handlers, 293
connectKeyPress, 350
connectListBox, 165
connectListView, 170
connectMonthCalendar, 171
connectMonthCalendarEvent, 381
connectMove, 386
connectNotifyEvent, 388
connectPosChanged, 391
connectRadioButton, 172
connectReBarEvent, 392
connectResize, 397
connectResizing, 399
connectSizeMoveEnded, 410
connectStaticEvent, 412
connectStatusBarEvent, 415
connectTab, 173
connectToolBarEvent, 423
connectToolTipEvent, 428
connectTrackBar, 174
connectUpDown, 175
connectUpDownEvent, 457
constant methods, 68
constDir, 488
createBrush, 238
createFont, 542
createFontEx, 543
createPen, 550
createToolTip, 102
data attribute names, 157
data attribute problems, 161
data attributes, 156
deleteFont, 551
deleteObject, 551
disable, 500
disableControl, 126
disconnectKeyPress, 353
displaceBitmap, 241
display, 501
dlgHandle, 74
dlgID, 74
dlgUnit2pixel, 196
draw, 502
drawAngleArc, 551
drawArc, 552
drawBitmap, 242
drawLine, 553
drawPie, 553
drawPixel, 553
enable, 503
enableControl, 127
endAsyncExecution, 243
events
    ACTIVATE, 311
execute, 97
executeAsync, 243
extended window methods, 89
factorX, 493
factorY, 493
fillDrawing, 554
fillRect, 555
focusControl, 128
fontColor, 556
fontName, 76
fontToDC, 556
foregroundWindow, 503
freeDC, 557
get and set data methods, 156
getArcDirection, 557
getBitmapPosition, 246
getBitmapSize, 246
getBitmapSizeX, 247
getBitmapSizeY, 247
getBmpDisplacement, 248
getCheckBoxData, 176
getClientRect, 504

getControlData, 177
getControlHandle, 198
getControlID, 199
getControlText, 129
getData, 178
getDataAttribute, 179
getDataStem, 179
getDC, 557
getEditData, 180
getExStyleRaw, 504
getFont, 558
getID, 505
getListBoxData, 180
getListItemHeightPx, 249
getPixel, 558
getPos, 505
getRadioButtonData, 181
getRealPos, 506
getRealSize, 506
getRect, 507
getSelf, 197
getSize, 507
getStyleRaw, 508
getSysBrush, 559
getText, 509
getTextAlign, 559
getTextExtent, 560
getTextSizeDlg, 201
getTextSizeDu, 200
getTextSizePx, 509, 510
getTextSizeTitleBar, 202
hasKeyPressConnection, 355
help, 122
hide, 511
hideControl, 131
hideControlFast, 131
hideFast, 512
hScrollPos, 561
hwnd, 493
initAutoDetection, 181
initCode, 494
initDialog, 99
installBitmapButton, 253
isEnabled, 512
isVisible, 513
loadBitmap, 561
menu methods, 195
move, 514
moveTo, 515
moveWindow, 517
new method, 70
newCheckBox, 103
newComboBox, 104
newDateTimePicker, 104
newEdit, 105
newGroupBox, 106
newListBox, 106
newListView, 107
newMonthCalendar, 107
newProgressBar, 108
newPushButton, 109
newRadioButton, 109
newReBar, 110
newScrollBar, 110
newStatic, 111
newStatusBar, 111
newTab, 112
newToolBar, 112
newToolTip, 113
newTrackBar, 114
newTreeView, 115
newUpDown, 116
noAutoDetection, 182
objectToDC, 562
ok, 123
opaqueText, 562
ownerDialog, 78
parseIncludeFile, 489, 489, 490
pixel2dlgUnit, 203
pixelCX, 494
pixelCY, 495
popup, 255
popupAsChild, 257
preparing and running, 95
rectangle, 563
redrawClient, 520
redrawRect, 259
redrawWindowRect, 261
removeBitmap, 563
resize, 520
resizeTo, 522
screen2client, 523
screenToClient, 524
scroll, 564
scrollBitmapFromTo, 262
scrollButton, 263
scrollInControl, 264
scrollText, 266
sendMessage, 524, 525
sendMessageToControl, 207
sendMessageToControlH, 206
sendMessageToWindowH, 208
setArcDirection, 564
setBitmapPosition, 268
setCheckBoxData, 182
setComboBoxData, 183
setControlColor, 269
setControlData, 183

setControlSysColor, 270
setControlText, 137
setData, 184
setDataAttribute, 185
setDataStem, 186
setEditData, 187
setFont, 565
setForegroundWindow, 271
setGroup, 139
setHScrollPos, 565
setListBoxData, 188
setListColumnWidth, 272
setListColumnWidthPx, 273
setListItemHeightPx, 273
setListWidth, 274
setListWidthPx, 274
setRadioButtonData, 189
setRect, 526
setSBPos, 275
setStyleRaw, 528
setText, 529
setTextAlign, 566
setTitle, 530
setVScrollPos, 567
setWindowPos, 530
setWindowRect, 276
showFast, 534
sizeWindow, 534
standard event handlers, 121
tabToNext, 143
tabToPrevious, 144
title, 537
title=, 537
transparentText, 567
understanding data attributes, 156
update, 537
updateWindow, 538
validate, 124
vScrollPos, 567
windowRect, 539
write, 567
writeDirect, 570
writeToControl, 280
writeToWindow, 278
dialog object attributes, 73
dialog object class
getWindowText, 130
setWindowText, 141
DialogCcontrol class
data=, 842
DialogControl Class, 226
DialogControl class
assignFocus, 830
clearRect, 831
connectCharEvent, 832
connectFKeyPress, 835
connectKeyPress, 837
data method, 841
getTextSizeDlg, 844
group, 845
hasKeyPressConnection, 846
hDlg, 814
id, 814
oDlg, 815
redrawRect, 847
setColor, 847
setParent, 848
setSysColor, 850
setWindowTheme, 850
tabStop, 851
textSize, 852
useVersion, 853
usingUnicode, 854
usingVersion, 854
DialogControl object
disconnectKeyPress, 843
DialogExtensions
createBrush, 238
endAsyncExecution, 243
executeAsync, 243
getBitmapSizeX, 247
getBitmapSizeY, 247
setSBPos, 275
writeToControl, 280
writeToWindow, 278
DialogExtensions Class, 231
DialogExtensions class
clearControlRect, 235
clearRect, 236
dimBitmap, 240
drawBitmap, 242
getBitmapPosition, 246
getBitmapSize, 246
getBmpDisplacement, 248
getListItemHeightPx, 249
getListWidthPx, 250
installBitmapButton, 253
popup, 255
popupAsChild, 257
redrawRect, 259
redrawWindowRect, 261
scrollBitmapFromTo, 262
scrollButton, 263
scrollInControl, 264
scrollText, 266
setBitmapPosition, 268
setControlColor, 269
setControlSysColor, 270

setForegroundWindow, 271
setListColumnWidth, 272
setListColumnWidthPx, 273
setListItemHeightPx, 273
setListWidth, 274
setListWidthPx, 274
setWindowRect, 276
dimBitmap, 240
disable, 500
menu object, 1391
disable methods, 125
disableControl, 126
disableInternalResize, 931
disconnectKeyPress
dialog control object, 843
dialog object, 353
displaceBitmap, 241
display, 501
DlgArea class, 647
b, 650
cx, 651
cy, 651
h, 652
height, 651
hr, 653
l, 650
left, 650
margin, 651
r, 650
right, 653
t, 650
top, 650
w, 653
width, 650
wr, 654
x, 654
y, 654
DlgAreaU class, 656
correctionFactor, 658
lastError, 658
new, 657
noMove, 658
noMovePut, 659
noResize, 658
noResizePut, 659
resizable dialogs, 661
resize, 660
updateOnResize, 659
dlgData., 71
dlgHandle, 74
dlgID, 74
dlgTitle, 1681
dlgUnit2pixel, 196
DlgUtil class, 1593
and, 1594
comCtl32Version, 1595
errMsg, 1596
getGUID, 1597
getSystemMetrics, 1598
hiWord, 1599
loWord, 1599
makeLParam, 1600
makeWParam, 1601
or, 1602
screenArea, 1603
screenSize, 1603
shiftLeft, 1604
shiftRight, 1604
sHiWord, 1605
signed, 1605
signed32, 1606
sLoWord, 1606
sShiftLeft, 1607
sShiftRight, 1608
threadID, 1608
unsigned, 1609
unsigned32, 1610
version, 1611
windowFromPoint, 1612
doubleClickTime, 1545
dragDetect, 1564
dragHeight, 1645
dragWidth, 1646
draw, 502
draw methods
dialog object, 208
drawAngleArc, 551
drawArc, 552
drawBitmap, 242
drawButton, 243
drawLine, 553
drawPie, 553
drawPixel, 553
drawStage, 1070, 1351
DropHighlight, 1321
dropHighlighted
ListView class, 996
TreeView class, 1321
duplicate
ImageList class, 1480
dynamically resizable dialogs
DlgAreaU class, 661
DynamicDialog Class, 282

E

edit
ListView class, 996
TreeView class, 1321

Edit class, 926
  addStyle, 929
  canUndo, 930
  clearText, 930
  copyText, 931
  cutText, 931
  disableInternalResize, 931
  ensureCaretVisibility, 932
  events
    CHANGE, 344
    ERRSPACE, 344
    GOTFOCUS, 344
    HSCROLL, 344
    LOSTFOCUS, 344
    MAXTEXT, 344
    UPDATE, 344
    VSCROLL, 344
    WANTRETURN, 959
  firstVisibleLine, 933
  getCue, 933
  getLine, 934
  getMargins, 935
  getRect, 935
  getStyle, 936
  hideBalloon, 937
  isGrandchild, 937
  isModified, 939
  isSingleLine, 939
  lineFromIndex, 940
  lineIndex, 940
  lineLength, 941
  lines, 943
  lineScroll, 943
  margins, 944
  noContextMenu, 944
  passwordChar, 928
  pasteText, 945
  removeStyle, 945
  replaceSelText, 946
  replaceStyle, 947
  scrollCommand, 948
  select, 949
  selected, 950
  selection, 950
  setCue, 951
  setLimit, 952
  setMargins, 952
  setModified, 953
  setReadOnly, 954
  setRect, 954
  setTabStops, 955
  showBalloon, 956
  tab=, 958
  undo, 958
  wantReturn, 958
enable, 503
  menu object, 1391
enableControl, 127
enableOpenDropDown, 1708
enableWizButtons, 592
endAsyncExecution, 243
endEdit
  ListView class, 997
  TreeView class, 1322
endExecution, 572
endMonth, 1145
endNow, 1516
endVisualGroup, 1709
ensureCaretVisibility, 932
ensureVisible
  dialog object, 128
  ListView class, 997
  TreeView class, 1322
enumTools, 1272
equateTo, 1637
errMsg, 1596
ErrorDialog, 1529
event notification
  coding event handler examples, 295
  coding event handlers, 293
  connecting event handlers, 293
    common willReply argument, 299
  property sheet page event handlers, 612
EventNotification
  connectMonthCalendarEvent, 381
EventNotification class, 293
  addUserMsg, 301
  connectActivate, 310
  connectAllSBEvents, 404
  connectButtonEvent, 312
  connectComboBoxEvent, 317
  connectCommandEvents, 320
  connectDateTimePickerEvent, 324
  connectDraw, 337
  connectFKeyPress, 348
  connectHelp, 345
  connectKeyPress, 350
  connectMove, 386
  connectNotifyEvent, 388
  connectPosChanged, 391
  connectReBarEvent, 392
  connectResize, 397
  connectResizing, 399
  connectSizeMoveEnded, 410
  connectStatusBarEvent, 415
  connectToolBarEvent, 423
  connectToolTipEvent, 428
  connectUpDownEvent, 457

disconnectFKeyPress, 353
events
DRAW, 338
HELP, 346
KEYPRESS, 356
MOVE, 387
POSCHANGED, 392
RESIZE, 398
RESIZING, 401
SIZEMOVEENDED, 411
hasKeyPressConnection, 355
execute, 97
ControlDialog class, 572
InputBox class, 1501
PropertySheetDialog class, 593
SingleSelection class, 1525
TimedMessage class, 1527
executeAsync, 243
Expand, 1322

F

factorX, 493
factorY, 493
FilenameDialog, 1530
FileViewer.rex
example code, 145
fillDrawing, 554
fillRect, 555
find, 1323
ComboBox class, 887
ListBox class, 966
ListView class, 997
findComboEntry, 153
findListEntry, 191
findNearestXY, 998
findPartial, 998
FindWindow, 1612
firstVisible
ListView class, 999
TreeView class, 1324
firstVisibleLine, 933
fixFullRowColumns, 999
flags, 1297
focus
ListView class, 1000
Tab class, 1226
focusControl, 128
focused
ListView class, 1000
Tab class, 1226
font, 1071, 1351
fontColor, 556
fontName, 76
fontSize
dialog object, 78
fontToDC, 556
foregroundWindow, 503
forHitTest, 1293
forID, 1294
freeControlDC, 245
freeDC, 557
freeWindowDC, 245
fromArray, 1078
fromFiles
Image class, 1461
fromIDs
Image class, 1463
future directions, 37

G

get, 197
get and set data methods, 156
getAcceleration, 1358
getArcDirection, 557
getAsyncKeyState, 1578
getAutoConnectStatus, 1392
getBandBorders, 1161
getBandCount, 1161
getBandInfo, 1162
getBandMargins, 1162
getBarHeight, 1163
getBarInfo, 1163
getBase, 1359
getBitmapPosition, 246
getBitmapSize, 246
getBitmapSizeX, 247
getBitmapSizeY, 247
getBkColor, 1164
getBmpDisplacement, 248
getBorders, 1206
getBubbleSize, 1273
getBuddy, 1359
getButton, 1243
getButtonText, 1243
getButtonTextEx, 1244
getCalendarBorder, 1113
getCalendarCount, 1114
getCallD, 1114
getCapture, 1565
getCheck, 1001
getCheckBoxData, 176
getCheckButtonState, 1710
getCheckState
CheckBox class, 861
RadioButton class, 866
getClientRect, 504
getClipCursor, 1566
getColor, 1115

getColorScheme, 1164
getColumnCount
  ListView class, 1001
getColumnInfo, 1002
getColumnOrder
  ListView class, 1003
getColumnText, 1004
getCombBoxInfo, 889
getComboBoxData, 177
getComboEntry, 154
getComboItems, 154
getControlData, 177
getControlDC, 248
getControlHandle, 198
getControlID, 199
getControlItemState, 1710
getControlRect, 249
getControlState, 1711
getControlText, 129
getCount
  ImageList class, 1481
  Menu class, 1393
getCue
  ComboBox class, 888
  Edit class, 933
getCurrentComboIndex, 154
getCurrentListIndex, 191
getCurrentPageHwnd, 593
getCurrentSelection, 1731
getCurrentToolInfo, 1273
getCurrentView, 1116
getCursorPos, 1566
getData, 178
getDataAttribute, 179
getDataStem, 179
getDateTime, 911
getDayState, 1146
getDayStateBuffer, 1147
getDC, 557
getDelayTime, 1274
getDisabledImageList, 1244
getDisplayName, 1690
getEditBoxText, 1712
getEditControl, 890
getEditData, 180
getExStyleRaw, 504
getExtendedStyle, 1165
  ListView class, 1005
  ToolBar class, 1245
getExtendedStyleRaw, 1006
getFileName, 1732
getFileTypeIndex, 1733
getFirstDayOfWeek, 1116
getFirstVisible
  ComboBox class, 891
  ListBox class, 966
getFocus, 129
getFolder
  BrowseForFolder class, 1691
  CommonItemDialog class, 1733
  SimpleFolderBrowse class, 1772
getFont, 558
getFontName
  dialog object, 70
getFontSize
  dialog object, 70
getFullRange
  ProgressBar class, 1154
getFullRow, 1007
getGridInfo, 1117
getGUID, 1597
getHandle, 1393
getHelpID, 1394
getHorizontalExtent
  ComboBox class, 891
getHotImageList, 1246
getHoverTime, 1009
getIcon
  ImageList class, 1481
  Static class, 1200
  StatusBar class, 1207
getID, 505
  menu object, 1394
getIdealSize
  Button class, 869
  DateTimePicker class, 912
getImage, 870
  Image class, 1464
  ResourceImage class, 1489
  Static class, 1201
getImageList, 1166
  Button class, 870
  ListView class, 1009
  Tab class, 1227
  ToolBar class, 1246
  TreeView class, 1324
getImages
  ResourceImage class, 1491
getImageSize
  ImageList class, 1483
getInfo, 912
getItem, 1010
getItemData
  ListView class, 1012
  TreeView class, 1325
getItemHeight
  ComboBox class, 892
  TreeViewClass, 1326

getItemIDList, 1692
getItemInfo, 1013
getItemPos, 1014
getItemRect, 1015
  Tab class, 1227
  TreeView class, 1326
getItemState, 1395
getLine, 934
getLineStep, 1305
getListBoxData, 180
getListEntry, 191
getListItemHeight, 249
getListItemHeightPx, 249
getListItems, 192
getListWidth, 250
getListWidthPx, 250
getMargin, 1274
getMargins, 935
getMaxHeight, 1396
getMaxSelection, 1120
getMaxTipWidth, 1275
getMaxTodayWidth, 1120
getMenuBar
  dialog object, 195
getMenuHandle, 1397
getMinRect, 1120
getMinVisible, 892
getMonthCal, 914
getMonthCalColor, 915
getMonthCalFont, 916
getMonthCalStyle, 917
getMonthDelta, 1121
getMonthRange, 1122
getObject, 623
getPage, 594
getPageStep, 1305
getPalette, 1166
getParts, 1208
getPixel, 558
getPopup, 1397
getPos, 505
  ProgressBar class, 1154
getPosition, 1360
getPressedImageList, 1247
getRadioButtonData, 181
getRange, 1360
  DateTimePicker class, 918
  MonthCalendar class, 1122
getRealPos, 506
getRealSize, 506
getRect
  Edit class, 935
  ReBar class, 1167
  StatusBar class, 1209
  WindowBase class, 507
getResult
  CommonItemDialog class, 1734
  PropertySheetDialog class, 594
getResults, 1765
getRowCount, 1167
getRowHeight, 1168
getRValue, 1464, 1464, 1466
getSBPos, 250
getSBRange, 250
getSelectedControlItem, 1713
getSelectionRange, 1123
getSelf, 197
getSize, 507
getStateImage, 1327
getStyle, 936
  MonthCalendar class, 1124
getStyleRaw, 508
getSubitem
  ListView class, 1016
getSubitemRect, 1017
getSysBrush, 559
getSystemMetrics, 1598
getTabControl, 595
getText
  ComboBox class, 893
  ListBox class, 967
  menu object, 1398
  StatusBar class, 1209
  ToolTip class, 1275
  WindowBase class, 509
getTextAlign, 559
getTextColor, 1168
getTextExtent, 560
getTextLength, 1211
getTextMargin, 871
getTextSizeDlg
  dialog control object, 844
  dialog object, 201
getTextSizeDu, 200
getTextSizePx, 509
getTextSizeScreen, 510
getTextSizeTitleBar, 202
getTick, 1306
getTipBkColor, 1276
getTipText, 1212
getTipTextColor, 1276
getTitle, 1277
getToday, 1125
getToolCount, 1278
getToolInfo, 1278
getToolTips, 1169
  ListView class, 1018
  TreeView class, 1327

getType, 1399
getView, 1019
getWindowDC, 251
getWindowText, 130
gray, 1400
group, 845
GroupBox class, 862
    style=, 863
groupID, 1093

H

h, 652
handle
    Image class, 1469
    ImageList class, 1484
    ResourceImage class, 1493
hasCheckBoxes, 1019
hasCurrentTool, 1279
hasKeyPressConnection
    dialog control object, 846
    dialog object, 355
hasMenuBar
    dialog object, 196
hDlg, 814
header, 586
headerSubtitle, 614
headerTitle, 615
height
    DlgArea class, 651
    Size class, 1635
help
    dialog object, 122
    PropertySheetPage class, 623
hide, 511
hideBalloon, 937
hideControl, 131, 131
hideFast, 512
hideWindow, 132
hideWindowFast, 133
hilite, 1401
hint, 1682
Hit Test Keywords, 1541
hitTestInfo, 1169
    ListBox class, 967
    ListView class, 1020
    MonthCalendar class, 1125
    ToolTip class, 1279
    TreeViewClass, 1328
hiWord, 1599
hMenu, 1377
hr, 653
hScrollPos, 561
hwnd, 493
hwndToIndex, 595

I

id, 814, 1072, 1188, 1352
    Image class, 1467
idToIndex
    PropertySheetDialog class, 595
    ReBar class, 1170
Image class, 1459
imageIndex, 1093, 1104, 1189
imageList, 587
ImageList class, 1473
    getIcon, 1481
includeItem, 1770
incr, 1626
increase, 1516
incrStep, 1510
indent, 1094
Indent, 1329
Indent=, 1329
index, 1094
indexTocommand, 1247
indexToHwnd, 595
indexToID, 596
indexToPage, 596
InfoDialog, 1532
init
    CheckList class, 1499
    InputBox class, 1500
    ListChoice class, 1502
    MultiInputBox class, 1503
    TimedMessage class, 1525
initAutoDetection, 181
initCode, 494
initCOM
    BrowseForFolder class, 1694
    CommonItemDialog class, 1735
initDialog, 99, 597
initialStatusText, 1510
initialThread, 1683
initRange, 1306
initSelRange, 1307
initUpdateListView, 573
InputBox
    class, 1500
    function, 1532
inRect, 1627
insert
    ComboBox class, 893
    ListBox class, 968, 968
    ListView class, 1022
    Tab class, 1227
    TreeView class, 1330
insertBand, 1170
insertButton, 1248

insertColumn, 1023
insertColumnPX, 1025
insertComboEntry, 155
insertFullRow, 1026
insertItem, 1402
insertListEntry, 192
insertPage, 597
insertPopup, 1404
insertSeparator, 1406
insertSubItem, 1082
installAnimatedButton, 251
installBitmapButton, 253
IntegerBox
  class, 1501
  function, 1533
internal sorting
  ListView Class, 1076
  LvFullRow Class, 1076
interrupt
  Alerter class, 1521
  Interruptible class, 1523
interruptible
  Alerter class, 1522
  Interruptible class, 1523
Interruptible class, 1522
  interrupt, 1523
  interruptible, 1523
is32on64bit, 1615
is64bit, 1615
IsAncestor, 1332
isAssigned, 1446
isAtLeastVista, 1615
isAtLeastW2K, 1616
isAtLeastW2K3, 1616
isAtLeastWindows7, 1616
isAtLeastXP, 1616
isAttached, 1425
isButtonChecked, 1248
isButtonDown, 1567
isButtonEnabled, 1249
isButtonHidden, 1249
isButtonHighlighted, 1250
isButtonIndeterminate, 1250
isButtonPressed, 1251
isCanceled, 1521
isChecked
  CommonDialogCustomizations class, 1714
  ListView class, 1027
  menu object, 1407
isCommandItem, 1408
isComplete, 1439
isDialogActive, 99
isDisabled, 1408
isDropDown, 894
isDropDownList, 894
isDropDownOpen, 895
isEnabled, 512
  menu object, 1409
isGrandchild, 895, 937
isGrayed, 1410
isIndeterminate, 861
isMaximized, 133
isMinimized, 133
isModified, 939
isNull
  Image class, 1471
  ImageList class, 1484
  ResourceImage class, 1494
isPopup, 1410
isReleased, 1737
isSeparator, 1411
isServer2008, 1616
isServer2008R2, 1617
isSimple
  ComboBox class, 896
  StatusBar class, 1212
isSingleLine, 939
isSingleSelection, 969
isValidItemID, 1411
isValidMenu, 1413
isValidMenuHandle, 1413
isVisible, 513
isVista, 1617
isW2K, 1617
isW2K3, 1617
isWindows7, 1617
isWow64, 1617
isXP, 1618
isXP32, 1618
isXP64, 1618
item, 1072, 1083, 1105, 1353
itemData, 1073, 1095, 1353
  ReBarBandInfo class, 1189
  TbButton class, 1259
itemHeight, 969
itemHeight=, 969
itemHeightPx, 969
itemHeightPx=, 970
itemInfo
  ListView class, 1027
  Tab class, 1228
  TreeView class, 1332
itemPos, 1028
items
  ComboBox class, 897
  ListBox class, 970
  ListView class, 1029
  Tab class, 1229

TreeView class, 1333
itemsPerPage, 1029
itemState, 1029, 1095
itemStateMask, 1096
itemText, 1030, 1334
itemTextToMethodName, 1413

K

key2name, 1658
Keyboard class, 1577
getAsyncKeyState, 1578
killActive, 623

L

l, 650
last
ListView class, 1030
Tab class, 1229
lastError, 658
lastSelected, 1030
leaving, 124
left
DlgArea class, 650
Rect class, 1630
level, 1354
License, Common Public, 1798
License, Open Object Rexx, 1798
lineFromIndex, 940
lineIndex, 940
lineLength, 941
lines, 943
lineScroll, 943
list box methods, 189
listAddDirectory, 193
ListBox class, 961
add, 963
addDirectory, 963
columnWidth=, 964
columnWidthPx=, 964
delete, 964
deleteAll, 965
deSelectIndex, 965
deSelectRange, 965
events
DBLCLK, 359
ERRSPACE, 359
GOTFOCUS, 359
LOSTFOCUS, 359
SELCANCEL, 359
SELCHANGE, 359
find, 966
getFirstVisible, 966
getText, 967
hitTestInfo, 967
insert, 968
isSingleSelection, 969
itemHeight, 969
itemHeight=, 969
itemHeightPx, 969
itemHeightPx=, 970
items, 970
makeFirstVisible, 970
modify, 970
select, 971
selected, 971
selectedIndex, 971
selectedIndexes, 971
selectedItems, 972
selectIndex, 972
selectRange, 973
setTabulators, 973
setWidth, 974
setWidthPx, 974
width, 974, 975
ListChoice
class, 1502
function, 1533
listDrop, 193
ListView class, 976
add, 983
addExtendedStyle, 984
addFullRow, 986
addRow, 987
addRowFromArray, 988
columnWidthPX, 993
constant methods, 981
delete, 994
deleteAll, 994
deleteColumn, 994
deselectAll, 996
events
ACTIVATE, 374
BEGINDRAG, 364
BEGINEDIT, 365
BEGINRDRAG, 364
BEGINSCROLL, 367
CHANGED, 374
CHANGING, 374
CHECKBOXCHANGED, 368
CLICK, 369
COLUMNCLICK, 370
CUSTOMDRAW, 1066
DBLCLK, 369
DELETE, 374
DELETEALL, 374
ENDEDIT, 371
ENDSCROLL, 367

FOCUSCHANGED, 373
General, 374
GETINFOTIP, 375
INSERTED, 374
KEYDOWN, 377
KEYDOWNEX, 378
SELECTCHANGED, 380
SELECTFOCUSCHANGED, 380
firstVisible, 999
fixFullRowColumns, 999
getColumnInfo, 1002
getColumnText, 1004
getExtendedStyle, 1005
getFullRow, 1007, 1009
getItem, 1010
getItemData, 1012
getItemInfo, 1013
getItemPos, 1014
getItemRect, 1015
getSubitem, 1016
getSubitemRect, 1017
getToolTips, 1018
getView, 1019
hasCheckBoxes, 1019
hitTestInfo, 1020
insertColumn, 1023
insertColumnPX, 1025
insertFullRow, 1026
insertSubItem, 1082
isChecked, 1027
modifyColumnPX, 1032
modifyFullRow, 1033
modifyItem, 1034
modifySubitem, 1035
prependFullRow, 1038
removeItemData, 1040
setColumnWidth, 1048
setFullRowText, 1049
setItemData, 1053
setToolTips, 1057
setView, 1057
sortItems, 1059
stringWidthPX, 1062
load, 804
loadBitmap, 561
loadCursor, 1545
loadCursorFromFile, 1547
loadFrame, 806
loadImages, 1252
loadItems, 807
Locate, 1613
loWord, 1599
LvCustomDrawSimple class, 1066
clrText, 1069
clrTextBk, 1069
drawStage, 1070
font, 1071
id, 1072
item, 1072
itemData, 1073
new, 1069
reply, 1074
subItem, 1075
LvFullRow class, 1075
addSubItem, 1081
fromArray, 1078
item, 1083
new, 1079
removeSubItem, 1083
subItem, 1084
subItems, 1085
userData, 1080
LvItem class, 1085
columnFormats, 1091
columns, 1092
groupID, 1093
imageIndex, 1093
indent, 1094
index, 1094
itemData, 1095
itemState, 1095
itemStateMask, 1096
mask, 1097
new, 1086
overlayImageIndex, 1099
stateImageIndex, 1100
text, 1100
LvSubItem class, 1101
imageIndex, 1104
item, 1105
mask, 1105
new, 1102
subItem, 1106
text, 1107

M

makeDayStateBuffer, 1142
makeFirstVisible
ListBox class, 970
MakeFirstVisible
TreeView class, 1334
makeLParam, 1600
makeProminent, 1714
makeWParam, 1601
manageAtypicalTool, 1281
mapWindowPoints, 513
margin, 651
margins, 944

markButton, 1252
MarqueeMode, 1510
MarqueePause, 1511
mask, 1097, 1105, 1190
maximize, 134
maximizeBand, 1171
maxSize, 668
menu bar object, 1423
  attachTo, 1423
  detach, 1425
  isAttached, 1425
  redraw, 1425
  replace, 1426
Menu class, 1374
menu command event handlers, 1371
menu object, 1374
  check, 1378
  checkRadio, 1379
  command event connections, 1368
  command event handler, 1371
  command events, 1367
  command items, 1367
  connectAllCommandEvents, 1380
  connectCommandEvent
    class method, 1376
    instance method, 1381
  connectMenuEvent, 1382
  connectSomeCommandEvents, 1388
  deletePopup, 1389
  destroy, 1390
  disable, 1391
  enable, 1391
  getAutoConnectStatus, 1392
  getCount, 1393
  getHandle, 1393
  getHelpID, 1394
  getID, 1394
  getItemState, 1395
  getMaxHeight, 1396
  getMenuHandle, 1397
  getPopup, 1397
  getText, 1398
  getType, 1399
  gray, 1400
  hilite, 1401
  hMenu, 1377
  insertItem, 1402
  insertPopup, 1404
  insertSeparator, 1406
  isChecked, 1407
  isCommandItem, 1408
  isDisabled, 1408
  isEnabled, 1409
  isGrayed, 1410
  isPopup, 1410
  isSeparator, 1411
  isValidItemID, 1411
  isValidMenu, 1413
  isValidMenuHandle, 1413
  itemTextToMethodName, 1413
  releaseMenuHandle, 1414
  removeItem, 1415
  removePopup, 1415
  removeSeparator, 1416
  setAutoConnection, 1417
  setHelpID, 1418
  setID, 1419
  setMaxHeight, 1420
  setText, 1421
  uncheck, 1421
  unhilite, 1422
  wID, 1378
menu object event
  CONTEXTMENU, 1384
  INITMENU, 1386
  INITMENUPOPUP, 1387
menuAnimation, 1646
MenuBar class, 1423
menuDropAlignment, 1644
menuFade, 1647
Menus, 1366
MessageDialog, 1533
methods for handles, sizes, and positions, 196
methods to query OS values, 196
minimize, 135
minimizeBand, 1172
minSize, 669
Modal dialog, 4
Modeless dialog, 4
modify, 1229
  ComboBox class, 897
  ListBox class, 970
  ListView class, 1031
  TreeView class, 1335
modifyColumn, 1032
modifyColumnPX, 1032
modifyFullRow, 1033
modifyItem, 1034
modifySubitem, 1035
MonthCalendar class, 1109
  addStyle, 1112
  date, 1111
  events
    GETDAYSTATE, 383
    RELEASED, 384
    SELCHANGED, 384
    SELECT, 385
    VIEWCHANGE, 386

getCalendarBorder, 1113
getCalendarCount, 1114
getCalID, 1114
getColor, 1115
getCurrentView, 1116
getFirstDayOfWeek, 1116
getGridInfo, 1117
getMaxSelection, 1120
getMaxTodayWidth, 1120
getMinRect, 1120
getMonthDelta, 1121
getMonthRange, 1122
getRange, 1122
getSelectionRange, 1123
getStyle, 1124
getToday, 1125
hitTestInfo, 1125
removeStyle, 1128
replaceStyle, 1129
setCalendarBorder, 1130
setCALID, 1131
setColor, 1132
setCurrentView, 1133
setDayState, 1134
setDayStateQuick, 1135
setFirstDayOfWeek, 1136
setMaxSelection, 1137
setMonthDelta, 1137
setRange, 1138
setSelectionRange, 1138
setToday, 1139
sizeRectToMin, 1139
Mouse class, 1541
appStarting, 1549
arrow, 1551
capture, 1551
clipCursor, 1552
connectEvent, 1553
cross, 1564
doubleClickTime, 1545
dragDetect, 1564
events
CAPTURECHANGED, 1557
LBUTTONDBLCLK, 1558
LBUTTONDOWN, 1558
LBUTTONUP, 1558
MBUTTONDBLCLK, 1558
MBUTTONDOWN, 1558
MBUTTONUP, 1558
MOUSEHOVER, 1558
MOUSELEAVE, 1561
MOUSEMOVE, 1558
MOUSEWHEEL, 1561
NCMOUSEHOVER, 1560
NCMOUSELEAVE, 1561
RBUTTONDBLCLK, 1558
RBUTTONDOWN, 1558
RBUTTONUP, 1558
getCapture, 1565
getClipCursor, 1566
getCursorPos, 1566
isButtonDown, 1567
loadCursor, 1545
loadCursorFromFile, 1547
new, 1544
no, 1568
releaseCapture, 1568
releaseClipCursor, 1569
restoreCursor, 1570
setCursor, 1571
setCursorPos, 1572
setDoubleClickTime, 1548
showCursor, 1573
swapButton, 1548
trackEvent, 1574
wait, 1577
mouseHoverHeight, 1647
mouseHoverTime, 1648
mouseHoverWidth, 1648
move, 514
moveBand, 1172
moveControl
dialog object, 254
moveItem
TreeView class, 1337
moveTo, 515
moveWindow, 517
msgHeight, 1512
msgText, 1512
MSSleep, 1614
MultiInputBox
class, 1503
function, 1538
MultiListChoice
class, 1504
function, 1538

N

new
Alerter class, 1520
ApplicationManager class, 1582
BinaryMenuBar class, 1427
BrowseForFolder class, 1680
CommonDialogEvents class, 1752
CommonItemDialog class, 1724
DayState class, 1140
DayStates class, 1144
dialog object, 70

DlgArea class, 649
DlgAreaU class, 657
Image class, 1467
ImageList class, 1476
LvCustomDrawSimple class, 1069
LvFullRow class, 1079
LvItem class, 1086
LvSubItem class, 1102
Mouse class, 1544
OpenFileDialog class, 1765
Point class, 1622
PopupMenu class, 1442
ProgressDialog class, 1507
RcControlDialog class, 575
RcDialog class, 687
RcPSPDialog class, 637
ReBarBandInfo class, 1181
Rect class, 1629
ResControlDialog class, 577
ResDialog class, 583, 685
ResourceImage class, 1488
ResPSPDialog class, 640
SaveFileDialog class, 1768
ScriptMenuBar class, 1429
ShellItemFilter class, 1770
SingleSelection class, 1524
Size class, 1634
SystemMenu class, 1454
TbButton class, 1257
ToolInfo class, 1295
TvCustomDrawSimple class, 1350
UserControlDialog class, 579
UserDialog, 692
UserMenuBar class, 1432
UserPSPDialog class, 643
Window class, 1661
newCheckBox, 103
newComboBox, 104
newDateTimePicker, 104
newEdit, 105
newGroupBox, 106
newListBox, 106
newListView, 107
newMonthCalendar, 107
newProgressBar, 108
newPushButton, 109
newRadioButton, 109
newReBar, 110
newScrollBar, 110
newStatic, 111
newStatusBar, 111
newTab, 112
newToolBar, 112
newToolRect, 1283
newToolTip, 113
newTrackBar, 114
newTreeView, 115
newUpDown, 116
next
ListView class, 1036
TreeView class, 1337
nextLeft, 1036
nextRight, 1037
nextSelected, 1037
NextVisible, 1338
no, 1568
noAutoDetection, 182
noContextMenu, 944
noMaxSize, 680
noMinSize, 681
noMove, 658
noMovePut, 659
noMsg, 1513
noResize, 658
noResizePut, 659
noStatus, 1513
Notices, 1796

O

objectToDC, 562
oDlg, 815
ok, 123
onButtonClicked, 1753
onCheckButtonToggled, 1754
onControlActivating, 1754
onFileOk, 1755
onFolderChange, 1756
onFolderChanging, 1757
onHelp, 1758
onItemSelected, 1758
onOverwrite, 1760
onSelectionChange, 1761
onShareViolation, 1761
onTypeChange, 1762
ooDialog executable, 41
command line options, 41
configure file associations, 45
overview, 41
program launcher, 42
ooDialog public routines, 1528
ooRexx License, 1798
opaqueText, 562
Open Object Rexx License, 1798
openDropDown, 898
OpenFileDialog class, 1763
getResults, 1765
new, 1765
options

BrowseForFolder class, 1685
CommonItemDialog class, 1725
or, 1602
OS class, 1614
is32on64bit, 1615
is64bit, 1615
isAtLeastVista, 1615
isAtLeastW2K, 1616
isAtLeastW2K3, 1616
isAtLeastWindows7, 1616
isAtLeastXP, 1616
isServer2008, 1616
isServer2008R2, 1617
isVista, 1617
isW2K, 1617
isW2K3, 1617
isWindows7, 1617
isWow64, 1617
isXP, 1618
isXP32, 1618
isXP64, 1618
settingChanged, 1618
shellChangeNotify, 1619
overlayImageIndex, 1099
overview, 1
owner, 1687
ownerDialog, 78

P

pageCreate, 624
pagedTab, 681
pageID, 615
pageNumber, 616
pages, 588
pageTitle, 616
pageToIndex, 598
Parent, 1338
parseIncludeFile, 489
PasswordBox
class, 1504
function, 1539
passwordChar, 928
pasteText, 945
pixel2dlgUnit, 203
pixelCX, 494
pixelCY, 495
PlainBaseDialog Class, 461
Play, 1621
Point class, 1621
+, 1624
-, 1624
copy, 1625
decr, 1626
incr, 1626
inRect, 1627
string, 1628
x, 1623
y, 1623
pop, 1284
popup
dialog object class, 255
rPropertySheetDialog class, 598
popUp
ToolTip class, 1284
popupAsChild
DialogExtensions class, 257
PropertySheetDialog class, 599
PopupMenu class, 1439
assignTo, 1443
connectContextMenu
class method, 1441
instance method, 1444
isAssigned, 1446
new, 1442
show, 1446
track, 1449
PopupMenu Event
CONTEXTMENU, 1439
pos, 1307
pos=, 1308
position, 1195
posRectangle, 1230
prepare4nItems, 1037
preparing and running dialogs, 95
prependFullRow, 1038
pressButton, 600
previous
ListView class, 1038
TreeView class, 1338
previousSelected, 1039
PreviousVisible, 1339
progress dialog implementation, 1505
ProgressBar class, 1151
ProgressDialog class, 1505
absoluteWidth, 1508
Alerter class, 1520
begin, 1515
canCancel, 1509
captionText, 1509
complete, 1515
constant methods, 1507
endNow, 1516
increase, 1516
incrStep, 1510
initialStatusText, 1510
marqueeMode, 1510
marqueePause, 1511
msgHeight, 1512

msgText, 1512
new, 1507
noMsg, 1513
noStatus, 1513
rangeMax, 1514
rangeMin, 1514
setInterruptible, 1517
updateStatus, 1518
updateStatusText, 1519
width, 1514
Property Sheet dialogs, 571
Property sheet dialogs
PropertySheetDialog class, 580
Property Sheet Pages, 609
PropertySheetPage class, 610
RcPSPDialog class, 637
ResPSPDialog class, 640
UserPSPDialog class, 643
PropertySheetDialog class, 580
addPage, 590
appIcon, 585
apply, 591
cancelToClose, 591
caption, 586
changed, 591
enableWizButtons, 592
execute, 593
getCurrentPageHwnd, 593
getPage, 594
getResult, 594
getTabControl, 595
header, 586
hwndToIndex, 595
idToIndex, 595
imageList, 587
indexToHwnd, 595
indexToID, 596
indexToPage, 596
initDialog, 597
insertPage, 597
pages, 588
pageToIndex, 598
popup, 598
popupAsChild, 599
pressButton, 600
querySiblings, 600
rebootSystem, 601
recalcPageSizes, 601
removePage, 602
resources, 588
restartWindows, 602
setButtonText, 603
setCurSel, 604
setCurSelByID, 604
setFinishText, 605
setHeaderSubTitle, 605
setHeaderTitle, 605
setNextText, 606
setTitle, 606
setWizButtons, 607
showWizButtons, 608
startPage, 589
unchanged, 609
watermark, 589
PropertySheetPage class, 610
apply, 621
cx, 613
cy, 614
event handlers, 612
getObject, 623
headerSubtitle, 614
headerTitle, 615
help, 623
killActive, 623
pageCreate, 624
pageID, 615
pageNumber, 616
pageTitle, 616
propSheet, 617
queryCancel, 625
queryFromSibling, 626
queryInitialFocus, 627
reset, 628
resources, 618
setActive, 628
setSize, 629
tabIcon, 619
translateAccelerator, 630
validate, 634
validatePage, 636
wantAccelerators, 620
wantGetObject, 620
wasActivated, 620
wizBack, 632
wizFinish, 633
wizNext, 634
propSheet, 617
public routines
AskDialog, 1528
CheckList, 1528
ErrorDialog, 1529
FileNameDialog, 1530
FindWindow, 1612
InfoDialog, 1532
InputBox, 1532
IntegerBox, 1533
ListChoice, 1533
Locate, 1613

MessageDialog, 1533
MSSleep, 1614
MultiInputBox, 1538
MultiListChoice, 1538
PasswordBox, 1539
Play, 1621
ScreenSize, 1633
SimpleFolderBrowse, 1773
SingleSelection, 1539
TimedMessage, 1540
WinTimer, 1670
push, 872
Push buttons, 867
pushChevron, 1173
putMonth, 1148
putYear, 1149

Q

queryCancel, 625
queryFromSibling, 626
queryInitialFocus, 627
querySiblings, 600
quickDayStateBuffer, 1144

R

r, 650
RadioButton class, 863
check, 865
checked, 866
checkInGroup, 865
getCheckState, 866
uncheck, 867
range
ScrollBar class, 1195
Trackbar class, 1308
rangeMax, 1514
rangeMin, 1514
RcControlDialog Class, 574
RcDialog class, 686
RcPSPDialog class, 637
ReBar class, 1159
deleteBand, 1160
events
AUTOBREAK, 394
NCHITTEST, 395
RELEASEDCAPTURE, 396
getBandBorders, 1161
getBandCount, 1161
getBandInfo, 1162
getBandMargins, 1162
getBarHeight, 1163
getBarInfo, 1163
getBkColor, 1164
getColorScheme, 1164
getExtendedStyle, 1165
getImageList, 1166
getPalette, 1166
getRect, 1167
getRowCount, 1167
getRowHeight, 1168
getTextColor, 1168
getToolTips, 1169
hitTestInfo, 1169
idToIndex, 1170
insertBand, 1170
maximizeBand, 1171
minimizeBand, 1172
moveBand, 1172
pushChevron, 1173
setBandInfo, 1173
setBandWidth, 1174
setBarInfo, 1174
setBkColor, 1175
setColorScheme, 1175
setExtendedStyle, 1176
setImageList, 1176
setPalette, 1177
setParent, 1178
setTextColor, 1178
setToolTips, 1179
setWindowTheme, 1179
showBand, 1180
sizeToRect, 1180
ReBarBandInfo class, 1181
bitmapBack, 1182
chevronLocation, 1182
chevronState, 1183
child, 1183
clrBack, 1184
clrFore, 1184
cx, 1185
cxHeader, 1185
cxIdeal, 1186
cxMinChild, 1186
cyChild, 1187
cyIntegral, 1187
cyMaxChild, 1187
cyMinChild, 1188
id, 1188
imageIndex, 1189
itemData, 1189
mask, 1190
new, 1181
style, 1190
text, 1191
rebootSystem, 601
recalcPageSizes, 601

rect, 1298
Rect class, 1628
  bottom, 1630
  copy, 1631
  left, 1630
  right, 1631
  string, 1632
  top, 1631
rectangle, 563
redraw, 519
  menu bar object, 1425
redrawClient, 520
redrawControl, 259
redrawItems, 1039
redrawRect
  dialog control object, 847
  dialog object, 259
redrawWindow, 261
redrawWindowRect, 261
release, 1737
  Image class, 1470
  ImageList class, 1485
  ResourceImage class, 1494
releaseCapture, 1568
releaseClipCursor, 1569
releaseCOM
  BrowseForFolder class, 1695
  CommonItemDialog class, 1739
releaseItemIDList, 1697
releaseMenuHandle, 1414
remove
  ImageList class, 1485
removeAll
  ImageList class, 1486
removeAllControlItems, 1715
removeBitmap, 563
removeControlItem, 1716
removeExtendedStyle, 1040
removeFullColor, 898
removeItem, 1415
removeItemData
  ListView class, 1040
  TreeViewClass, 1339
removePage, 602
removePopup, 1415
removeSeparator, 1416
removeStyle
  Edit class, 945
  ListView class, 1042
  MonthCalendar class, 1128
removeSubItem, 1083
replace, 1426
replaceExtendedStyle, 1042
replaceSelText, 946
replaceStyle
  Edit class, 947
  ListView class, 1044
  MonthCalendar class, 1129
reply, 1074, 1355
requiredOS, 1587
requiredWindowSize, 1230
ResControlDialog Class, 577
ResDialog class, 685
reset, 628
resizable dialogs, 647
resize
  dialog control object, 520
  dialog object, 520
  DlgAreaU class, 660
  WindowBase, 520
Resize a dialog, 695, 698
resizeControl, 262
resizeTo, 522
Resizing class
  defaultLeft, 675
  defineSizing, 679
ResizingAdmin Class, 663
  basics, 664
ResizingAdmin class
  controlBottom, 669
  controlLeft, 670
  controlRight, 671
  controlSizing, 672
  controlTop, 673
  defaultBottom, 674
  defaultRight, 676
  defaultSizing, 677
  defaultTop, 678
  events
    EXITSIZEMOVE, 684
  maxSize, 668
  minSize, 669
  noMaxSize, 680
  noMinSize, 681
  pagedTab, 681
  useDefaultSizing, 682
  wantSizeEnded, 683
resolveNumericID, 489
resolveSymbolicID, 490
resource, 1299
Resource file dialogs, 685
  RcDialog class, 686
  ResDialog class, 685
ResourceImage class, 1487
resources
  PropertySheetDialog class, 588
  PropertySheetPage class, 618
Resources, 1459

ResourceUtils Class, 487
ResourceUtils class
  constDir, 488
  parseIncludeFile, 489
  resolveNumericID, 489
  resolveSymbolicID, 490
ResPSPDialog class, 640
restartWindows, 602
restore, 136
restoreCursor, 1570
revert, 1456
rexxHwnd, 1299
rexxID, 1300
rgb, 225
right
  DlgArea class, 653
  Rect class, 1631
root, 1340, 1688
rows, 1231

S

SaveFileDialog class, 1766
  new, 1768
  setSaveAsItem, 1768
screen2client, 523
screenArea, 1603
screenSize, 1603
ScreenSize, 1633
screenToClient, 524
ScriptMenuBar class, 1429
  new, 1429
scroll
  ListView class, 1045
  WindowExtensions class, 564
scroll bar event constants, 1193
scroll bar methods
  dialog object, 204
ScrollBar class, 1192
  determinePosition, 1194
  events
    SCROLL, 408
  position, 1195
  range, 1195
  scroll bar event constants, 1193
  setPos, 1195
  setRange, 1196
scrollBitmapFromTo, 262
scrollButton, 263
scrollCommand, 948
scrollInControl, 264
scrollText, 266
select
  ComboBox class, 899
  Edit class, 949
  ListBox class, 971
  ListView class, 1046
  Tab class, 1231
  TreeView class, 1340
selected
  ComboBox class, 899
  Edit class, 950
  ListBox class, 971
  ListView class, 1046
  Tab class, 1232
  TreeView class, 1340
selectedIndex
  ComboBox class, 900
  ListBox class, 971
  Tab class, 1232
selectedIndexes, 971
selectedItems
  ListBox class, 972
  ListView class, 1046
selectIndex
  ComboBox class, 900
  ListBox class, 972
  Tab class, 1232
selection, 950
selectRange, 973
selRange, 1308
sendMessage, 524
sendMessageHandle, 525
sendMessageToControl, 205, 207
sendMessageToControlH, 206
sendMessageToWindowH, 208
setAcceleration, 1361
setActive, 628
setArcDirection, 564
setAutoConnection, 1417
setBandInfo, 1173
setBandWidth, 1174
setBarInfo, 1174
setBase, 1362
setBitmapPosition, 268
setBitmapSize, 1253
setBkColor
  ReBar class, 1175
  StatusBar class, 1213
setBuddy, 1362
setButtonText
  PropertySheetDialog class, 603
  ToolBar class, 1253
setCalendarBorder, 1130
setCALID, 1131
setCancelButtonLabel, 1740
setCheckBoxData, 182
setCheckButtonState, 1716
setClientGuid, 1741

setColor, 847, 1132
setColorScheme, 1175
setColumnOrder
  ListView class, 1046
setColumnWidth, 1048, 1048
setComboBoxData, 183
setControlColor, 269
setControlData, 183
setControlFont, 270
setControlItemState, 1717
setControlItemText, 1718
setControlLabel, 1718
setControlState, 1719
setControlSysColor, 270
setControlText, 137
setCue
  ComboBox class, 900
  Edit class, 951
setCurrentComboIndex, 155
setCurrentListIndex, 193
setCurrentView, 1133
setCurSel, 604
setCurSelByID, 604
setCursor, 1571
setCursorPos, 1572
setData, 184
setDataAttribute, 185
setDataStem, 186
setDateTime, 920
setDayState, 1134
setDayStateQuick, 1135
setDefaultExtension, 1742
setDefaultFolder, 1743
setDefaultFont, 72
setDefaults, 1588
setDelayTime, 1284
setDisabledImageList, 1254
setDlgFont
  dialog object, 100
setDoubleClickTime, 1548
setEditBoxText, 1720
setEditData, 187
setEditSelection, 901
setExtendedStyle
  ReBar class, 1176
  ToolBar class, 1254
setFileName, 1744
setFileNameLabel, 1744
setFileTypeIndex, 1745
setFileTypes, 1746
setFilter, 1747
setFinishText, 605
setFirstDayOfWeek, 1136
setFocus, 137
setFocusToWindow, 138
setFolder, 1747
setFont, 565
setForegroundWindow, 271
setFormat, 920
setFullColor, 902
setFullRange
  ProgressBar class, 1155
setFullRowText, 1049
setGroup, 139
setHeaderSubTitle, 605
setHeaderTitle, 605
setHelpID, 1418
setHorizontalExtent
  ComboBox class, 903
setHotImageList, 1255
setHoverTime, 1050
setHScrollPos, 565
setIcon
  StatiC class, 1202
  StatusBar class, 1214
setID, 1419
setImage, 873
  Static class, 1202
setImageList, 873, 1255
  ListView class, 1051
  ReBar class, 1176
  Tab class, 1233
  TreeView class, 1341
setIndeterminate, 862
setInterruptible, 1517
setItemData
  ListView class, 1053
  TreeView class, 1342
setItemHeight
  ComboBox class, 904
  TreeViewClass, 1343
setItemPos, 1054
setItemSize, 1234
setItemState, 1055
setItemText, 1056
setLimit, 952
setLineStep, 1308
setListBoxData, 188
setListColumnWidth, 272
setListColumnWidthPx, 273
setListItemHeight, 273
setListItemHeightPx, 273
setListTabulators, 194
setListWidth, 274
setListWidthPx, 274
setMargin, 1285
setMargins, 952
setMarquee

ProgressBar class, 1156
setMax, 1309
setMaxHeight, 1420
setMaxSelection, 1137
setMaxTipWidth, 1286
setMin, 1309
setMinHeight, 1215
setMinVisible, 905
setModified, 953
setMonthCalColor, 921
setMonthCalFont, 922
setMonthCalStyle, 922
setMonthDelta, 1137
setNextText, 606
setOkButtonLabel, 1748
setPadding, 1235
setPageStep, 1310
setPalette, 1177
setParent
DialogControl class, 848
ReBar class, 1178
setParts, 1215
setPos
ProgressBar class, 1157
ScrollBar class, 1195
Trackbar class, 1310
setPosition, 1363
setPressedImageList, 1256
setRadioButtonData, 189
setRange, 1364
DateTimePicker class, 925
MonthCalendar class, 1138
ScrollBar class, 1196
setReadOnly, 954
setRect
Edit class, 954
WindowBase class, 526
setSaveAsItem, 1768
setSBPos, 275
setSBRange, 275
setSelectedControlItem, 1721
setSelectionRange, 1138
setSelEnd, 1311
setSelStart, 1311
setSize
PropertySheetPage class, 629
Tab class, 1235
setStateImage, 1343
setStep, 1157
setStyleRaw, 528
setSysColor, 850
setTabStop, 139
setTabStops
Edit class, 955
setTabulators, 973
setText, 529, 1216
menu object, 1421
setTextAlign, 566
setTextColor, 1178
setTextMargin, 876
setTickAt, 1312
setTickFrequency, 1312
settingChanged, 1618
setTipBkColor, 1287
setTipText, 1217
setTipTextColor, 1287
setTitle, 1749
PropertySheetDialog class, 606
ToolTip class, 1288
WindowBase class, 530
setToday, 1139
setToolInfo, 1289
setToolTips, 1179
ListView class, 1057
TreeView class, 1344
setView, 1057
setVScrollPos, 567
setWidth, 974
setWidthPx, 974
setWindowPos, 530
setWindowRect, 276
setWindowText, 141
setWindowTheme, 850, 1179, 1290
setWizButtons, 607
shell objects, 1672
shellChangeNotify, 1619
ShellItemFilter class, 1769
includeItem, 1770
new, 1770
shiftLeft, 1604
shiftRight, 1604
sHiWord, 1605
show, 1750
dialog control, 533
dialog object, 141
PopupMenu class, 1446
showBalloon, 956
showBand, 1180
showControl, 142
showControlFast, 143
showCursor, 1573
showFast, 534
showWindow, 143
showWindowFast, 143
showWizButtons, 608
signed, 1605
signed32, 1606
simple, 1218

SimpleFolderBrowse
  Special Folder Names, 1675
SimpleFolderBrowse class, 1771
  getFolder, 1772
SimpleFolderBrowse routine, 1773
SingleSelection
  function, 1539
  standard dialog, 1523
SingleSelection class
  execute, 1525
  new, 1524
Size class, 1633
  comparison operators, 1636
  equateTo, 1637
  height, 1635
  string, 1637
  width, 1635
sizeRectToMin, 1139
sizeToRect, 1180
sizeWindow, 534
sizeX, 496
sizeY, 497
sLoWord, 1606
SM, 1638
SM class
  cMouseButtons, 1639
  cxCursor, 1639
  cxDrag, 1639
  cxFixedFrame, 1640
  cxIcon, 1640
  cxScreen, 1640
  cxSize, 1641
  cxSmIcon, 1641
  cxVscroll, 1641
  cyCaption, 1641
  cyCursor, 1642
  cyDrag, 1642
  cyFixedFrame, 1642
  cyHscroll, 1643
  cyIcon, 1643
  cyMenu, 1643
  cyScreen, 1643
  cySize, 1644
  cySmIcon, 1644
  menuDropAlignment, 1644
smallSpacing, 1058
snapToGrid, 1058
SortChildren, 1344
sortChildrenCB, 1345
sortItems, 1059
spacing, 1062
SPI class, 1644
  dragHeight, 1645
  dragWidth, 1646
  menuAnimation, 1646
  menuFade, 1647
  mouseHoverHeight, 1647
  mouseHoverTime, 1648
  mouseHoverWidth, 1648
  updateFlag, 1649
  wheelScrollLines, 1650
  WHEEL_PAGESCROLL, 1645
  workArea, 1650
srcDir, 1582
sShiftLeft, 1607
sShiftRight, 1608
standard dialogs, 1498
  CheckList, 1499
  InputBox, 1500
  IntegerBox, 1501
  ListChoice, 1502
  MultiInputBox, 1503
  MultiListChoice, 1504
  PasswordBox, 1504
  SingleSelection, 1523
  TimedMessage, 1525
standard event handler methods, 121
startDir, 1689
startMonth, 1146
startPage, 589
startVisualGroup, 1721
state, 1259
  Button class, 876
state=, 877
stateImageIndex, 1100
Static class, 1197
  events
    CLICKED, 414
    DBLCLK, 414
    DISABLE, 414
    ENABLE, 414
  getIcon, 1200
  getImage, 1201
  setIcon, 1202
  setImage, 1202
StatusBar class, 1204
  events
    CLICK, 417
    DBLCLK, 417
    General, 417
    RCLICK, 417
    RDBLCLK, 417
    SIMPLEMODECHANGE, 418
  getBorders, 1206
  getIcon, 1207
  getParts, 1208
  getRect, 1209
  getText, 1209

getTextLength, 1211
getTipText, 1212
isSimple, 1212
setBkColor, 1213
setIcon, 1214
setMinHeight, 1215
setParts, 1215
setText, 1216
setTipText, 1217
simple, 1218
step, 1157
string
Point class, 1628
Rect class, 1632
Size class, 1637
stringWidth, 1062
stringWidthPX, 1062
style
ReBarBandInfo class, 1190
TbButton class, 1259
style=, 878
GroupBox class, 863
subItem, 1075, 1084, 1106
subItems, 1085
swapButton, 1548
systemErrorCode
Image class, 1472
ResourceImage class, 1495
SystemMenu class, 1452
command event handler, 1372
connectAllCommandEvents, 1454
connectCommandEvent, 1455
connectSomeCommandEvents, 1457
constant methods, 1453
revert, 1456

T

t, 650
Tab class, 1220
addFullSeq, 1222
addSequence, 1223
adjustToRectangle, 1223
calcDisplayRect, 1224
calcWindowRect, 1225
delete, 1225
deleteAll, 1226
events
KEYDOWN, 420
SELCHANGE, 421
SELCHANGING, 422
focus, 1226
focused, 1226
getImageList, 1227
getItemRect, 1227
insert, 1227
itemInfo, 1228
items, 1229
last, 1229
modify, 1229
posRectangle, 1230
requiredWindowSize, 1230
rows, 1231
select, 1231
selected, 1232
selectedIndex, 1232
selectIndex, 1232
setImageList, 1233
setItemSize, 1234
setPadding, 1235
setSize, 1235
tab=
Edit class, 958
tabIcon, 619
tabStop, 851
tabToNext, 143
tabToPrevious, 144
TbButton class, 1256
assignBitmapID, 1260
bitmapID, 1258
cmdID, 1258
itemData, 1259
new, 1257
state, 1259
style, 1259
text, 1260
Template, Dialog, 9, 690
text, 1191
LvItem class, 1100
LvSubItem class, 1107
TbButton class, 1260
ToolInfo class, 1300
textBkColor, 1063
textBkColor=, 1063
textColor, 1063
textColor=, 1064
textSize, 852
The Dialog Object, 56
The Menu Object, 1374
The MenuBar Object, 1423
THICKFRAME, 695, 698
threadID, 1608
tiledBackgroundBitmap, 150
TimedMessage
class, 1525
function, 1540
title, 537
title=, 537
Toggle, 1347

ToolBar class, 1237
  addBitmap, 1238
  addButtons, 1238
  addString, 1239
  autoSize, 1240
  buttonCount, 1240
  changeBitmap, 1241
  checkButton, 1241
  commandToIndex, 1242
  customize, 1242
  events
    BEGINADJUST, 426
    CLICK, 427
    CUSTHELP, 426
    DBLCLK, 427
    ENDJUST, 426
    General, 426, 427
    RCLICK, 427
    RDBLCLK, 427
    TOOLBARCHANGE, 426
  getButton, 1243
  getButtonText, 1243
  getButtonTextEx, 1244
  getDisabledImageList, 1244
  getExtendedStyle, 1245
  getHotImageList, 1246
  getImageList, 1246
  getPressedImageList, 1247
  indexTocommand, 1247
  insertButton, 1248
  isButtonChecked, 1248
  isButtonEnabled, 1249
  isButtonHidden, 1249
  isButtonHighlighted, 1250
  isButtonIndeterminate, 1250
  isButtonPressed, 1251
  loadImages, 1252
  markButton, 1252
  setBitmapSize, 1253
  setButtonText, 1253
  setDisabledImageList, 1254
  setExtendedStyle, 1254
  setHotImageList, 1255
  setImageList, 1255
  setPressedImageList, 1256
ToolInfo class, 1292
  flags, 1297
  forHitTest, 1293
  forID, 1294
  new, 1295
  rect, 1298
  resource, 1299
  rexxHwnd, 1299
  rexxID, 1300
  text, 1300
  userData, 1300
ToolTip class, 1262
  activate, 1265
  addTool, 1266
  addToolEx, 1268
  addToolRect, 1268
  adjustRect, 1270
  delTool, 1271
  enumTools, 1272
  events
    LINKCLICK, 429
    NEEDTEXT, 430
    POP, 431
    SHOW, 432
  getBubbleSize, 1273
  getCurrentToolInfo, 1273
  getDelayTime, 1274
  getMargin, 1274
  getMaxTipWidth, 1275
  getText, 1275
  getTipBkColor, 1276
  getTipTextColor, 1276
  getTitle, 1277
  getToolCount, 1278
  getToolInfo, 1278
  hasCurrentTool, 1279
  hitTest, 1279
  manageAtypicalTool, 1281
  newToolRect, 1283
  pop, 1284
  popUp, 1284
  setDelayTime, 1284
  setMargin, 1285
  setMaxTipWidth, 1286
  setTipBkColor, 1287
  setTipTextColor, 1287
  setTitle, 1288
  setToolInfo, 1289
  setWindowTheme, 1290
  tool identification, 1262
  trackActivate, 1290
  trackPosition, 1291
  update, 1292
  updateTipText, 1292
top
  DlgArea class, 650
  Rect class, 1631
toTheTop, 144
track, 1449
trackActivate, 1290
TrackBar class
  constants, 1303
  events

MOVE, 435
Trackbar class, 1302
trackEvent, 1574
trackPosition, 1291
translateAccelerator, 630
transparentText, 567
TreeView class, 1314
add, 1316
events
BEGINDRAG, 439
BEGINEDIT, 441
BEGINRDRAG, 439
DEFAULTEDIT, 443
DELETE, 443
ENDEDIT, 445
EXPANDED, 447
EXPANDING, 448
GETINFOTIP, 449
KEYDOWN, 450
KEYDOWNEX, 451
SELCHANGED, 453
SELCHANGING, 455
find, 1323
getItemData, 1325
getItemHeight, 1326
getItemRect, 1326
getStateImage, 1327
getToolTips, 1327
hitTestInfo, 1328
insert, 1330
itemInfo, 1332
itemText, 1334
modify, 1335
removeItemData, 1339
root, 1340
setItemData, 1342
setItemHeight, 1343
setStateImage, 1343
setToolTips, 1344
sortChildrenCB, 1345
TreeView Event
CUSTOMDRAW, 1348
TvCustomDrawSimple class, 1348
clrText, 1350
clrTextBk, 1350
drawStage, 1351
font, 1351
id, 1352
item, 1353
itemData, 1353
level, 1354
new, 1350
reply, 1355

U

unadvise, 1750
unchanged, 609
uncheck, 867
CommonDialogCustomizations class, 1722
ListView class, 1064
menu object, 1421
uncheckAll, 1065
understanding the create... methods, 699
undo, 958
unhilite, 1422
unsigned, 1609
unsigned32, 1610
update
ToolTip class, 1292
WindowBase class, 537
updateFlag, 1649
updateItem, 1065
updateOnResize, 659
updateStatus, 1518
updateStatusText, 1519
updateTipText, 1292
updateWindow, 538
UpDown class, 1356
deltaPosReply, 1357
events
DELTAPOS, 458
getAcceleration, 1358
getBase, 1359
getBuddy, 1359
getPosition, 1360
getRange, 1360
setAcceleration, 1361
setBase, 1362
setBuddy, 1362
setPosition, 1363
setRange, 1364
useDefaultSizing, 682
useGlobalConstDir, 1590
usePathForHint, 1689
User Input, 1497
UserControlDialog Class, 578
userData
LvFullRow class, 1080
ToolInfo class, 1300
UserDialog class, 690
addIconResource, 693
Common Control Styles, 702
create, 694
create button controls, 703
create combo box controls, 735
create edit controls, 742
create... methods, 699

createBitmapButton, 703
createBlackFrame, 769
createBlackRect, 771
createCenter, 697
createCheckBox, 706
createCheckBoxGroup, 709
createCheckBoxStem, 712
createComboBox, 736
createComboBoxInput, 738
createDateTimePicker, 740
createEdit, 742
createEditInput, 745
createEditInputGroup, 748
createEditInputStem, 751
createEtchedFrame, 772
createEtchedHorizontal, 773
createEtchedVertical, 774
createGrayFrame, 775
createGrayRect, 776
createGroupBox, 715
createListBox, 757
createListView, 760
createMonthCalendar, 762
createOkCancelLeftBottom, 716
createOkCancelLeftTop, 717
createOkCancelRightBottom, 717
createOkCancelRightTop, 718
createPasswordEdit, 755
createProgressBar, 764
createPushButton, 718
createPushButtonGroup, 721
createPushButtonStem, 723
createRadioButton, 727
createRadioButtonGroup, 730
createRadioButtonStem, 732
createReBar, 765
createScrollBar, 768
createStatic, 777
createStaticFrame, 781
createStaticImage, 783
createStaticText, 785
createStatusBar, 791
createTab, 793
createToolBar, 795
createTrackbar, 797
createTreeView, 799
createUpDown, 802
createWhiteFrame, 789
createWhiteRect, 790
defineDialog, 804
load, 804
loadFrame, 806
loadItems, 807
new, 692
  understanding the create... methods, 699
userIcon, 1468
  Image class, 1468
UserMenuBar class, 1431
  addItem, 1433
  addPopup, 1435
  addSeparator, 1437
  complete, 1438
  isComplete, 1439
UserPSPDialog class, 643
useUnicode, 853
useVersion, 853
usingUnicode, 854
usingVersion, 854
Utility Classes, 1580

V

validate
  dialog object, 124
  IntegerBox function, 1502
  PropertySheetPage class, 634
validatePage, 636
value, 1141
version, 1611
view styles, 976
virtual key codes, 1652
VisibleItems, 1347
VK class, 1652
  key2name, 1658
vScrollPos, 567

W

w, 653
wait, 1577
wantAccelerators, 620
wantGetObject, 620
wantReturn, 958
wantSizeEnded, 683
wasActivated, 574, 620
watermark, 589
wheelScrollLines, 1650
WHEEL_PAGESCROLL, 1645
wID, 1378
width
  DlgArea class, 650
  ListBox class, 974
  ProgressDialog class, 1514
  Size class, 1635
widthPx, 975
Window class, 1659
WindowBase
  clear, 498
  client2screen, 499

clientRect, 499
clientToScreen, 500
disable, 500
display, 501
draw, 502
enable, 503
factorX, 493
factorY, 493
foregroundWindow, 503
getClientRect, 504
getExStyleRaw, 504
getID, 505
getPos, 505
getRealPos, 506
getRealSize, 506
getRect, 507
getSize, 507
getStyleRaw, 508
getTextSizePx, 509, 510
hide, 511
hideFast, 512
hwnd, 493
initCode, 494
isEnabled, 512
isVisible, 513
move, 514
moveTo, 515
moveWindow, 517
pixelCX, 494
pixelCY, 495
redrawClient, 520
resize, 520
resizeTo, 522
screen2client, 523
screenToClient, 524
sendMessage, 524, 525
setRect, 526
setStyleRaw, 528
setText, 529
setWindowPos, 530
show, 533
showFast, 534
sizeWindow, 534
sizeX, 496
sizeY, 497
title, 537
title=, 537
windowRect, 539
WindowBase Class, 491
WindowBase class
childWindowFromPoint, 497
getText, 509
mapWindowPoints, 513
setTitle, 530
update, 537
updateWindow, 538
WindowExtensions Class, 539
WindowExtensions class
createBrush, 541
createFont, 542
createFontEx, 543
createPen, 550
deleteFont, 551
deleteObject, 551
drawAngleArc, 551
drawArc, 552
drawLine, 553
drawPie, 553
drawPixel, 553
fillDrawing, 554
fillRect, 555
fontColor, 556
fontToDC, 556
freeDC, 557
getArcDirection, 557
getDC, 557
getFont, 558
getPixel, 558
getSysBrush, 559
getTextAlign, 559
getTextExtent, 560
hScrollPos, 561
loadBitmap, 561
objectToDC, 562
opaqueText, 562
rectangle, 563
removeBitmap, 563
scroll, 564
setArcDirection, 564
setFont, 565
setHScrollPos, 565
setTextAlign, 566
setVScrollPos, 567
transparentText, 567
vScrollPos, 567
write, 567
writeDirect, 570
windowFromPoint, 1612
windowRect, 539
WinTimer, 1670
wizBack, 632
wizFinish, 633
wizNext, 634
workArea, 1650
wr, 654
write, 567
writeDirect, 570
writeToControl, 280

writeToWindow, 278

X

x
  DlgArea class, 654
  Point class, 1623

Y

y
  DlgArea class, 654
  Point class, 1623