ooRexx Documentation 5.0.0

Open Object Rexx

nCurses Class Library Reference

ooRexx Documentation 5.0.0 Open Object Rexx
nCurses Class Library Reference
Edition 2022.12.22

Author	W. David Ashley
Author	Rony G. Flatscher
Author	Mark Hessling
Author	Rick McGuire
Author	Lee Peedin
Author	Oliver Sims
Author	Erich Steinböck
Author	Jon Wolfers

Preface

This book describes the Open Object Rexx™ ncurses Function Library and Classes.

This book is intended for people who plan to develop applications using Rexx and ncurses. Its users range from the novice, who might have experience in some programming language but no Rexx or ncurses experience, to the experienced application developer, who might have had some experience with Object Rexx and ncurses.

This book is a reference rather than a tutorial. It assumes you are already familiar with object-oriented programming concepts.

Descriptions include the use and syntax of the language and explain how the language processor "interprets" the language as a program is running.

1. Document Conventions

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

1.1. 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.

1.2. 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.

2. How to Read the Syntax Diagrams

Throughout this book, syntax is described using the structure defined below.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

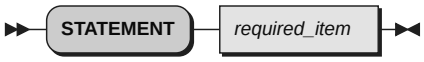
  The ▶▶— symbol indicates the beginning of a statement.

  The —··· symbol indicates that the statement syntax is continued on the next line.

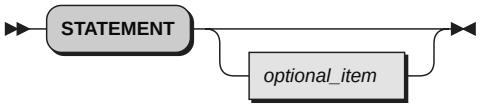
  The ···— symbol indicates that a statement is continued from the previous line.

  The —▶◀ symbol indicates the end of a statement.

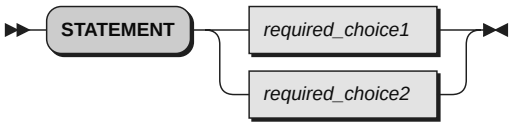
- Required items appear on the horizontal line (the main path).



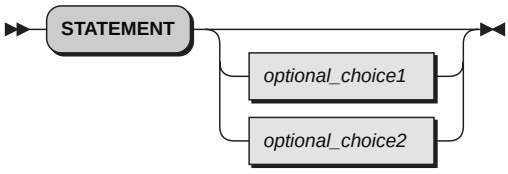
- Optional items appear below the main path.



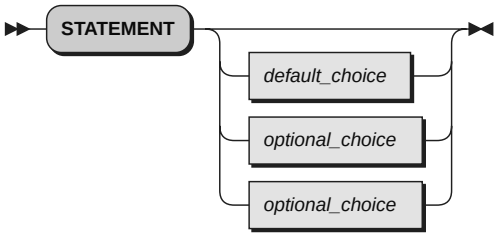
- If you can choose from two or more items, they appear vertically, in a stack. If you must choose one of the items, one item of the stack appears on the main path.



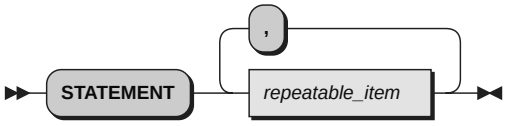
- If choosing one of the items is optional, the entire stack appears below the main path.



- If one of the items is the default, it is usually the topmost item of the stack of items below the main path.



- A path returning to the left above the main line indicates an item that can be repeated.



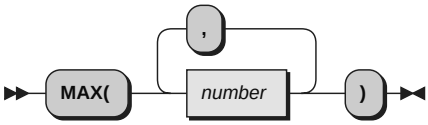
  A repeat path above a stack indicates that you can repeat the items in the stack.

- A pointed rectangle around an item indicates that the item is a fragment, a part of the syntax diagram that appears in greater detail below the main diagram.

- Keywords appear in uppercase (for example, **SIGNAL**). They must be spelled exactly as shown but you can type them in upper, lower, or mixed case. Variables appear in all lowercase letters (for example, *index*). They represent user-supplied names or values.

- If punctuation marks, parentheses, arithmetic operators, or such symbols are shown, you must enter them as part of the syntax.

The following example shows how the syntax is described:



3. 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, in no particular order of preference, are listed below.

3.1. The Open Object Rexx SourceForge Site

Open Object Rexx utilizes SourceForge to house its source repositories, mailing lists and other project features at *https://sourceforge.net/projects/oorexx*. ooRexx uses the Developer and User mailing lists at *https://sourceforge.net/p/oorexx/mailman* for discussions concerning ooRexx. The ooRexx user is most likely to get timely replies from one of these mailing lists.

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

The Developer Mailing List
: Subscribe to the oorexx-devel mailing list at *https://lists.sourceforge.net/lists/listinfo/oorexx-devel* to discuss ooRexx project development activities and future interpreter enhancements. You can find its archive of past messages at *http://sourceforge.net/mailarchive/forum.php?forum_name=oorexx-devel*.

The Users Mailing List
: Subscribe to the oorexx-users mailing list at *https://lists.sourceforge.net/lists/listinfo/oorexx-users* to discuss how to use ooRexx. It also supports a historical archive of past messages.

The Announcements Mailing List
: Subscribe to the oorexx-announce mailing list at *https://lists.sourceforge.net/lists/listinfo/oorexx-announce* to receive announcments of significant ooRexx project events.

The Bug Mailing List
: Subscribe to the oorexx-bugs mailing list at *https://lists.sourceforge.net/lists/listinfo/oorexx-bugs* to monitor changes in the ooRexx bug tracking system.

Bug Reports
: You can view ooRexx bug reports at *https://sourceforge.net/p/oorexx/bugs*. To be able to create new bug reports, you will need to first register for a SourceForge userid at *https://sourceforge.net/user/registration*. When reporting a bug, please try to provide as much information as possible to help developers determine the cause of the issue. Sample program code 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 *https://sourceforge.net/p/oorexx/documentation*. 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 new suggest ooRexx features or enhancements at *https://sourceforge.net/p/oorexx/feature-requests*.

Patch Reports
: If you create an enhancement patch for ooRexx please post the patch at *https://sourceforge.net/p/oorexx/patches*. 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 at *https://sourceforge.net/p/oorexx/bugs* 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 at *https://sourceforge.net/p/oorexx/discussion*. There are currently three forums available: Help, Developers and Open Discussion. In addition, you can monitor the forums via email.

3.2. The Rexx Language Association Mailing List

The Rexx Language Association maintains a forum at *http://www.rexxla.org/forum.html*.

3.3. comp.lang.rexx Newsgroup

The comp.lang.rexx newsgroup at *https://groups.google.com/forum/#!forum/comp.lang.rexx* is a good place to obtain help from many individuals within the Rexx community. You can obtain help on Open Object Rexx and other Rexx interpreters and tools.

4. Related Information

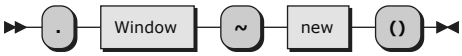
See also: *Open Object Rexx: Reference*

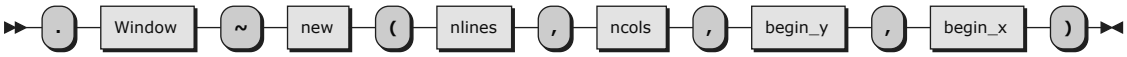
Window Class Method Reference

1.1. New - Class Method

Init the standard screen or create a new window.

Syntax:





Arguments:

The first invocation of this method creates the standard screen and must pass zero arguments. Subsequest invocations create standard windows and must pass four arguments.

nlines
: Number of lines.

ncols
: Number of columns.

begin_y
: Y position start position.

begin_x
: X position start position.

Returns:

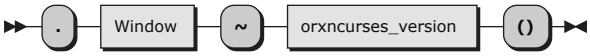
Zero.

> Example 1.1. Creating new windows
>
> ```
> /* create the standard screen */
> scr = .Window~new()
>
> /* create a new window */
> win = .Window~new(10, 20, 5, 10)
> ```

1.2. Orxncurses_Version - Class Methos

Return the Orxnncurses_Version string.

Syntax:



Arguments:

None.

Returns:

The OrxnCurses library version string.

1.3. SetBase - Class Method

Set whether or not the library uses one-based indexes or zero-based indexes. The default is one-based.

Syntax:

Arguments:

base
: 1 = one-based, 0 = zero-based

Returns:

The current or new base.

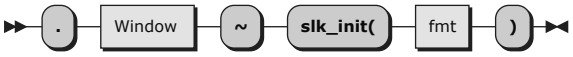
> Example 1.2. Set base (one or zero based)

```
.window~setbase(0)
scr = .window~new()

scr~move(0, 0)
scr~addch('*')
scr~refresh()
scr~napms(500)
```

1.4. Slk_init - Class Method

Initialize for soft label. This method must be invoked prior to creating the standard screen.

Syntax:



Arguments:

fmt
: Line format. Valid values are 0, 1, 2 and 3.

Returns:

Zero.

> Example 1.3. Set a soft label

```
left = 0
center = 1
right = 2

.window~slk_init(0)
```

```
scr = .window~new()

scr~slk_set(1, 'Help!', left)
scr~slk_set(2, 'File', left)
scr~slk_set(3, 'Print', left)
scr~slk_set(4, 'Text', center)
scr~slk_set(5, 'Edit', center)
scr~slk_set(6, 'Quick', right)
scr~slk_set(7, 'Conf', right)
scr~slk_set(8, 'Change', right)
scr~slk_refresh()
```

1.5. Acs_map

Return an ACS character.

Syntax:

Arguments:

str
: Character.

Returns:

ACS character.

1.6. Addch

Add a single character to the screen.

Syntax:

Arguments:

ch
: Character to be added. This is a single character, not a numeric value.

Returns:

Either ERR (-1) or OK (0).

Example 1.4. Add a character to the screen

```
scr = .window~new()

p = .pad~new(200, wide)

strm = .stream~new(filename)
strm~open('read')
eof = .false
call on notready name eof
```

```
ch = strm~charin()
do while \eof
   p~addch(ch)
   ch = strm~charin()
   end
strm~close()
```

1.7. Addchnstr

Add a chtype character string to the screen.

Syntax:

Arguments:

str
: Chtype string be added. A chtype is a four-byte sequence.

n
: Number of chtype characters to add.

Returns:

Either ERR (-1) or OK (0).

1.8. Addchstr

Add a chtype character string to the screen.

Syntax:

Arguments:

str
: Chtype string be added. A chtype is a four-byte sequence. The string can contain any number of chtypes.

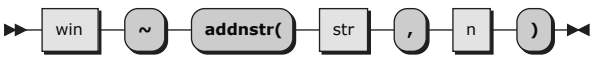
Returns:

Either ERR (-1) or OK (0).

1.9. Addnstr

Add a character string to the screen.

Syntax:



Arguments:

str
: String to be added.

n
: Number of characters to add.

Returns:

Either ERR (-1) or OK (0).

1.10. Addstr

Add a character string to the screen.

Syntax:

Arguments:

str
: String to be added.

Returns:

Either ERR (-1) or OK (0).

> Example 1.5. Add a string to the screen
>
> ```
> scr = .window~new()
> scr~addstr('Goodbye, cruel C Programming!')
> ```

1.11. Assume_default_colors

Set default colors.

Syntax:

Arguments:

fg
: Forground color.

bg
: Background color.

Returns:

Either ERR (-1) or OK (0).

1.12. Attroff

Turn off one or more attributes.

Syntax:

Arguments:

attr
: Attribute(s).

Returns:

Either ERR (-1) or OK (0).

Example 1.6. Turn off one or more attributes

```
scr = .window~new()
scr~attron(scr~A_BOLD)

scr~attroff(scr~A_BOLD)
```

1.13. Attron

Turn on one or more attributes.

Syntax:

Arguments:

attr
: Attribute(s).

Returns:

Either ERR (-1) or OK (0).

Example 1.7. Turn on one or more attributes

```
scr = .window~new()
scr~attron(scr~A_BOLD)
```

1.14. Attrset

Turn on one or more attributes.

Syntax:

Arguments:

attr
: Attribute(s).

Returns:

Either ERR (-1) or OK (0).

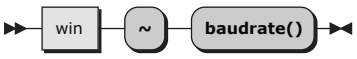
Example 1.8. Set one or more attributes

```
lf = .window~ASCII_LF~d2c()
bs = .window~ASCII_BS~d2c()

text = .array~of('Do', 'you', 'find', 'this', 'silly?')

scr = .window~new()
do a = 1 to text~items()
   do b = 1 to text~items()
      if b = a then scr~attrset(scr~A_BOLD + scr~A_UNDERLINE)
      scr~addstr(text[b])
      if b = a then scr~attroff(scr~A_BOLD + scr~A_UNDERLINE)
      scr~addch(' ')
      end
   scr~addstr(bs || lf)
   end
```

1.15. Baudrate

Return the terminal baud rate.

Syntax:



Arguments:

None.

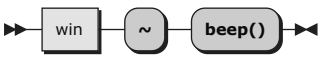
Returns:

Either ERR (-1) or OK (0).

1.16. Beep

Beep the terminal.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

Example 1.9. Beep the terminal

```
scr = .window~new()

scr~addstr('Attention!' || lf)
scr~beep()
```

1.17. Bkgd

Set backgroung attributes for the whole screen.

Syntax:

Arguments:

attr
: Attribute(s).

Returns:

Either ERR (-1) or OK (0).

Example 1.10. Set the background attribute for the whole screen

```
scr = .window~new()
scr~start_color()
scr~init_pair(1, scr~COLOR_WHITE, scr~COLOR_BLUE)

scr~bkgd(scr~COLOR_PAIR(1))
```

1.18. Bkgdset

Set backgroung attributes for the next output text.

Syntax:

Arguments:

attr
: Attribute(s).

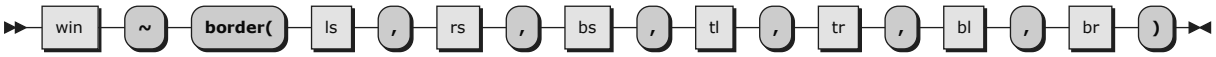
Returns:

Either ERR (-1) or OK (0).

1.19. Border

Set window border.

Syntax:



Arguments:

ls
: Left side character.

rs
: Right side character.

ts
: Top side character.

bs
: Bottom side character.

tl
: Top/Left character.

tr
: Top/Right character.

bl
: Bottom/Left character.

br
: Bottom/Right character.

Returns:

Either ERR (-1) or OK (0).

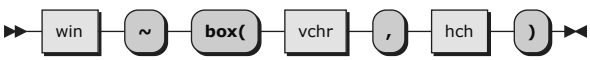
> Example 1.11. Set a window border

```
scr = .window~new()

scr~border('ba'~x2d(), 'ba'~x2d(), 'cd'~x2d(), 'cd'~x2d(),,
           'c9'~x2d(), 'bb'~x2d(), 'c8'~x2d(), 'bc'~x2d())
```

1.20. Box

Draw a box around the edges of a window.

Syntax:



Arguments:

vch
: Vertical character.

hch
: Horizontal character.

Returns:

Either ERR (-1) or OK (0).

Example 1.12. Draw a box arounf a window

```
scr = .window~new()

scr~box('*'~c2d(), '*'~c2d())
```

1.21. Can_change_color

Returns whether or not a terminal can changes its color set.

Syntax:

Arguments:

None.

Returns:

One or zero.

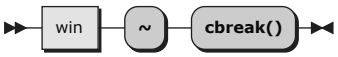
Example 1.13. Determine if terminal hardware can change its color set

```
scr = .window~new()
scr~start_color
if \scr~can_change_color() then do
   scr~addstr('This probably won''t work, but anyway:' || lf)
   end
```

1.22. Cbreak

Activates cbreak (buffering) mode.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

1.23. Chgat

Change attributes on the window.

Syntax:

Arguments:

n
: Number of character positions.

attr
: The attribute.

color
: The COLOR_PAIR number.

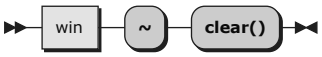
Returns:

Either ERR (-1) or OK (0).

1.24. Clear

Clear the window.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

> Example 1.14. Clear the screen
>
> ```
> scr = .window~new()
>
> scr~clear()
> ```

1.25. Clearok

Force a repaint of the window on the next refresh call.

Syntax:

Arguments:

bf
: Boolean on/off

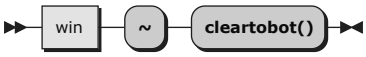
Returns:

Either ERR (-1) or OK (0).

1.26. Clrtobot

Clear the window from the current cursor position.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

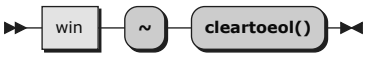
> Example 1.15. Clear the window from the current cursor position

```
scr = .window~new()

scr~move(6, 21)
scr~clrtobot()
```

1.27. Clrtoeol

Clear the line from the current cursor position.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

> Example 1.16. Clear the line from the current cursor position

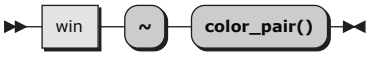
```
scr = .window~new()
scr~noecho()
scr~keypad(.true)

mmask = scr~mousemask(scr~ALL_MOUSE_EVENTS)

do forever
   ch = scr~getch()
   if ch = scr~KEY_MOUSE then do
      mort = scr~getmouse()
      scr~move(1, 1)
      scr~clrtoeol()
      scr~addstr(mort~y || '/' || mort~x)
```

```
      scr~refresh()
      end
   if ch = lf then leave
   end
```

1.28. Color_pair

Returns the number of possible COLOR_PAIR attributes.

Syntax:



Arguments:

None.

Returns:

Possible color pairs attributes.

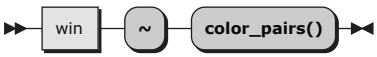
Example 1.17. Get the number of possible COLOR_PAIR attributes

```
scr = .window~new()

scr~start_color
scr~init_pair(1, scr~COLOR_WHITE, scr~COLOR_BLUE)
scr~bkgd(scr~color_pair(1))
```

1.29. Color_pairs

Returns the number of possible COLOR_PAIRS.

Syntax:



Arguments:

None.

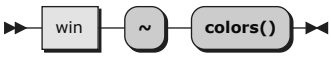
Returns:

Possible number of color pairs.

1.30. Colors

Returns the number of possible COLORs.

Syntax:



Arguments:

None.

Returns:

Number of colors.

1.31. Color_set

Sets foreground and background text color.

Syntax:

Arguments:

num
: COLOR_PAIR number.

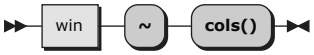
Returns:

Either ERR (-1) or OK (0).

1.32. Cols

Returns the number of columns on the stdscr.

Syntax:



Arguments:

None.

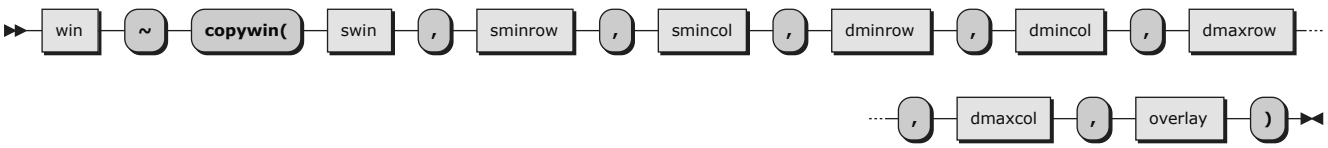
Returns:

Number of columns.

1.33. Copywin

Copy a rectangle from one window to self.

Syntax:



Arguments:

swin
: Source window.

sminrow
: Source min row number.

smincol
: Source min col number.

dminrow
: Destination min row number.

dmincol
: Destination min col number.

dmaxrow
: Destination max row number.

dmaxcol
: Destination max col number.

overlay
: Boolean yes/no to overlay destination text.

Returns:

Either ERR (-1) or OK (0).

Example 1.18. Copy a window to self

```
scr = .window~new()

maxyx = scr~getmaxyx()
parse var maxyx maxy maxx .
halfy = (maxy / 2)~format(, 0)
halfx = (maxx / 2)~format(, 0)

top = .window~new(halfy, halfx, 1, 1)
bottom = .window~new(halfy, halfx, halfy + 1, halfx + 1)

top~addstr(text1)
top~refresh()
bottom~addstr(text2)
bottom~refresh()

bottom~getch()

retc = bottom~copywin(top, 1, 1, 1, 1, 5, 13, .false)
```

1.34. Curs_set

Control the cursor visibility.

Syntax:

Arguments:

vis
: Visibility boolean.

Returns:

Either ERR (-1) or OK (0).

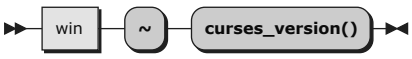
Example 1.19. Set cursor visibility

```
scr = .window~new()

-- turn off cursor
scr~curs_set(0)
-- turn the cursor on
scr~curs_set(1)
-- turn the cursor very on
scr~curs_set(2)
```

1.35. Curses_version

Return the nCurses version string.

Syntax:



Arguments:

None.

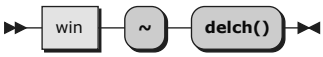
Returns:

Version string.

1.36. Delch

Delete the character under the cursor and slide remaining characters on the line one position to the left.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

Example 1.20. Delete the character under the cursor

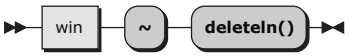
```
scr = .window~new()

scr~move(3, 26)
do c = 1 to 11
   scr~delch()
```

```
   end
```

1.37. Deleteln

Delete the line under the cursor and slide remaining lines below the cursor up one line.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

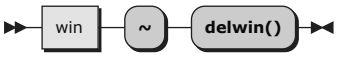
Example 1.21. Delete the line under the cursor

```
scr = .window~new()

scr~move(2, 1)
scr~deleteln()
```

1.38. Delwin

Destroy a window.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

Example 1.22. Delete a window

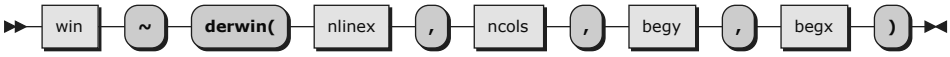
```
lf = .window~ASCII_LF~d2c()

scr = .window~new()

p = .pad~new(50, 100)

scr~addstr("New pad created")
scr~refresh()
scr~getch()

if p~delwin() = scr~OK then do
   scr~addstr("...and now it's gone!" "" lf)
   end
```

```
else do
   scr~addstr("...and now it's still there!" "" lf)
   end
scr~refresh()
```

1.39. Derwin

Create a derrived window from self.

Syntax:



Arguments:

nlines
: Number of lines.

ncols
: Number of cols.

begy
: Beginning Y line.

begx
: Beginning X column.

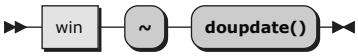
Returns:

A Window instance.

1.40. Doupdate

Update the terminal.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

Example 1.23. Update the terminal

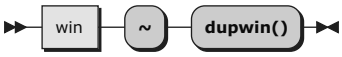
```
filename = 'Readme.txt'
tall = 24
wide = 19
spacer = 5

scr = .window~new()
p = .pad~new(200, wide + 1)
```

```
s1 = p~subpad(tall, wide + 1, 1, 1)
s2 = p~subpad(tall, wide + 1, tall + 1, 1)
s3 = p~subpad(tall, wide + 1, (2 * tall) + 1, 1)

strm = .stream~new(filename)
strm~open('read')
eof = .false
call on notready name eof

ch = strm~charin()
do while \eof
   p~addch(ch)
   ch = strm~charin()
   end
strm~close()

s1~pnoutrefresh(1, 1, 1, 1, tall, wide + 1)
s2~pnoutrefresh(1, 1, 1, wide + spacer + 1, tall, (wide * 2) + spacer)
s3~pnoutrefresh(1, 1, 1, (wide * 2) + (spacer * 2), tall, (wide *3) + (spacer * 2))
scr~doupdate()
```

1.41. Dupwin

Duplicate a window (self).

Syntax:



Arguments:

None.

Returns:

A Window instance.

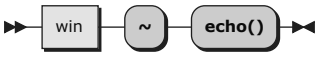
Example 1.24. Duplicate a window

```
numeric digits 12
lf = .window~ASCII_LF~d2c()

scr = .window~new()

/* Build window & wait */
fred = .window~new(0,0,1,1)
fred~addstr("This is the original window, Fred." || lf)
fred~refresh()
fred~getch()

/* Create and show barney */
barney = fred~dupwin()
```

1.42. Echo

Turn on echo.

Syntax:



Arguments:

None.

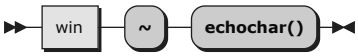
Returns:

Either ERR (-1) or OK (0).

1.43. Echochar

Echo one character.

Syntax:



Arguments:

None.

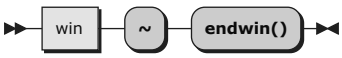
Returns:

Either ERR (-1) or OK (0).

1.44. Endwin

End nCurses formatting.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

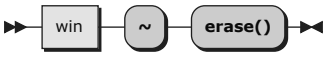
> Example 1.25. End nCurses formating

```
scr = .window~new()

scr~endwin()
```

1.45. Erase

Erase the window.

Syntax:



Arguments:

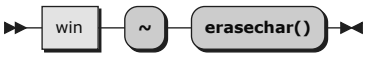
None.

Returns:

Either ERR (-1) or OK (0).

1.46. Erasechar

Syntax:



Return the terminal's erase char.

Arguments:

None.

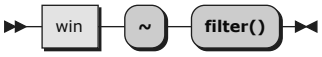
Returns:

Either ERR (-1) or OK (0).

1.47. Filter

Restrict output to a single line.

Syntax:



Arguments:

None.

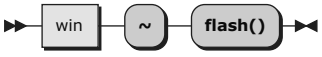
Returns:

Either ERR (-1) or OK (0).

1.48. Flash

Briefly flash the screen.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

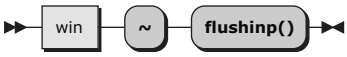
Example 1.26. Flash the screen

```
scr = .window~new()

scr~addstr('I said ATTENTION!')
scr~flash()
```

1.49. Flushinp

Flush the input queue.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

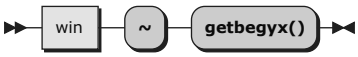
Example 1.27. Flash the input queue

```
scr = .window~new()

scr~flushinp()
```

1.50. Getbegyx

Get the y & x screen coordinate for the top left corner of the window relative to the stdscr.

Syntax:



Arguments:

None.

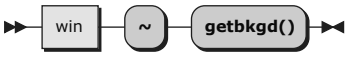
Returns:

A string in the form "y x".

1.51. Getbkgd

Get the background attribute for the the window.

Syntax:



Arguments:

None.

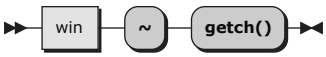
Returns:

Attribute (a numeric value).

1.52. Getch

Get character from the keyboard.

Syntax:



Arguments:

None.

Returns:

A one character string.

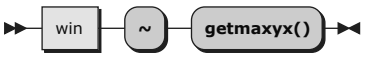
> Example 1.28. Get a character from the keyboard

```
scr = .window~new()

char = scr~getch()
```

1.53. Getmaxyx

Get the width and height of a window.

Syntax:



Arguments:

None.

Returns:

A string in the form "maxy maxx".

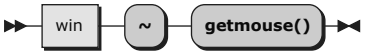
> Example 1.29. Get the width and height of the screen

```
scr = .window~new()

maxyx = scr~getmaxyx()
parse var maxyx maxy maxx .
```

1.54. Getmouse

Get a mouse event.

Syntax:



Arguments:

None.

Returns:

An instance of the Mevent class.

> Example 1.30. Get a mouse event

```
scr = .window~new()
scr~noecho()
scr~keypad(.true)

mmask = scr~mousemask(scr~ALL_MOUSE_EVENTS)

do forever
   ch = scr~getch()
   if ch = scr~KEY_MOUSE then do
      mort = scr~getmouse()
      scr~move(1, 1)
      scr~clrtoeol()
      scr~addstr(mort~y || '/' || mort~x)
      scr~refresh()
      end
   if ch = lf then leave
   end
```

1.55. Getnstr

Get a string from the terminal.

Syntax:

Arguments:

n
: Number of characters to get.

Returns:

A string.

> Example 1.31. Get a string from the terminal

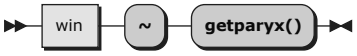
```
scr = .window~new()
```

```
scr~mvaddstr(4, 11, 'Enter your name: ')
scr~refresh()
name = scr~getnstr(45)
scr~mvaddstr(6, 7, 'Enter your password: ')
scr~refresh()
scr~noecho()
password = scr~getnstr(8)
```

1.56. Getparyx

Get the width and height of a window relative to the parent window.

Syntax:



Arguments:

None.

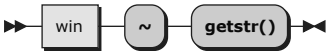
Returns:

A string in the form "maxy maxx".

1.57. Getstr

Get a string from the terminal.

Syntax:



Arguments:

None.

Returns:

A character string (maximum of 1023 characters).

1.58. Getwin

Create a new window from a file.

Syntax:

Arguments:

filename
: File name containing the window information.

Returns:

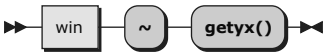
A Window instance.

> Example 1.32. Create a new window from a file
>
> ```
> scr = .window~new()
>
> win = scr~getwin('window.dat')
> win~refresh()
> ```

1.59. Getyx

Get the cursor location from a window.

Syntax:



Arguments:

None.

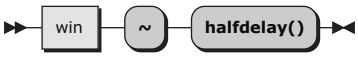
Returns:

A string in the form "y x".

1.60. Halfdelay

Simalar to the cbreak method.

Syntax:



Arguments:

tenths
: Tenths of a second.

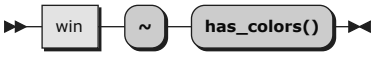
Returns:

Either ERR (-1) or OK (0).

1.61. Has_colors

Determine if the terminal has color support.

Syntax:



Arguments:

None.

Returns:

One or zero.

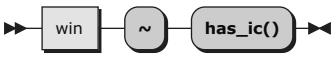
Example 1.33. Determine if the terminal has color support

```
scr = .window~new()

if \scr~has_colors() then do
   scr~endwin()
   say 'Terminal cannot do colors.'
   end
```

1.62. Has_ic

Determine if the terminal has insert character support.

Syntax:



Arguments:

None.

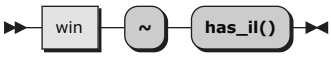
Returns:

One or zero.

1.63. Has_il

Determine if the terminal has insert line support.

Syntax:



Arguments:

None.

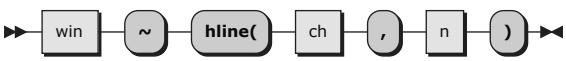
Returns:

One or zero.

1.64. Hline

Draw a horizontal line.

Syntax:



Arguments:

ch
: Character type (chtype)

n
: Number of characters to draw.

Returns:

Either ERR (-1) or OK (0).

Example 1.34. Draw a horizontal line

```
scr = .window~new()

maxyx = scr~getmaxyx()
parse var maxyx maxy maxx .
x = 1
do y = 1 to maxy
   scr~move(y, x)
   scr~hline(0, maxx - x + 1)
   scr~vline(0, maxy - y + 1)
   x += 2
   end
```

1.65. Idcok

Use hardware to insert/delete characters.

Syntax:

Arguments:

bf
: Boolean on/off.

Returns:

Either ERR (-1) or OK (0).

1.66. Idlok

Use hardware to insert/delete lines.

Syntax:

Arguments:

bf
: Boolean on/off.

Returns:

Either ERR (-1) or OK (0).

1.67. Immedok

Use hardware to provide auto refresh.

Syntax:

Arguments:

bf
: Boolean on/off.

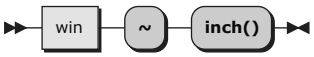
Returns:

Either ERR (-1) or OK (0).

1.68. Inch

Return the attr/char under the cursor.

Syntax:



Arguments:

None.

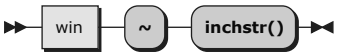
Returns:

A numeric containing the attribut and character.

1.69. Inchstr

Return the attr/char array.

Syntax:



Arguments:

None.

Returns:

A chtype array as a character string (maximum of 1023 bytes).

1.70. Inchnstr

Return the attr/char array.

Syntax:

Arguments:

n
: Number of attr/char to return.

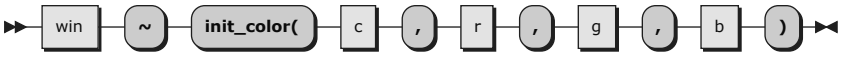
Returns:

A chtype array as a character string.

1.71. Init_color

Allow the user to redefine colors.

Syntax:



Arguments:

c
: Color number to change.

r
: Red value.

g
: Green value.

b
: Blue value.

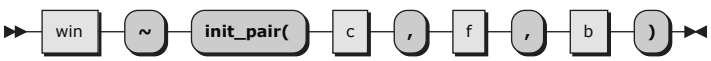
Returns:

Either ERR (-1) or OK (0).

1.72. Init_pair

Assign foreground an background colors to a color pair.

Syntax:



Arguments:

c
: COLOR_PAIR number to change.

f
: Foreground color.

b
: Background color.

Returns:

Either ERR (-1) or OK (0).

Example 1.35. Assign foreground and background colors

```
scr = .window~new()
scr~start_color()
scr~init_pair(1, scr~COLOR_WHITE, scr~COLOR_BLUE)
```

1.73. Innstr

Read a string from the terminal.

Syntax:

Arguments:

n
: Number of characters to read.

Returns:

A string.

1.74. Insch

Insert attr/char under the cursor.

Syntax:

Arguments:

ch
: chtype to be inserted.

Returns:

Either ERR (-1) or OK (0).

Example 1.36. Insert attr/character under the cursor

```
scr = .window~new()

do i = len to 1 by -1
   scr~move(6, 6)
   scr~insch(text~substr(i, 1))
   scr~refresh()
   scr~napms(100)
```

```
end
```

1.75. Insdelln

Insert/delete lines.

Syntax:

Arguments:

n
: Number of lines to insert/delete.

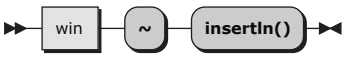
Returns:

Either ERR (-1) or OK (0).

1.76. Insertln

Insert one line.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

Example 1.37. Insert one line

```
scr = .window~new()

scr~move(2, 1)
scr~insertln()
```

1.77. Insnstr

Insert part of a string.

Syntax:

Arguments:

str
: String to insert.

n
: Number of characters to insert.

Returns:

Either ERR (-1) or OK (0).

1.78. Insstr

Insert a string.

Syntax:

Arguments:

str
: String to insert.

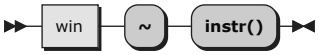
Returns:

Either ERR (-1) or OK (0).

1.79. Instr

Read a string from the terminal.

Syntax:



Arguments:

None.

Returns:

A string (maximum of 1023 bytes).

1.80. Intrflush

Turn on/off input queue flushing when a interrupt key is typed at the keyboard.

Syntax:

Arguments:

bf
: Boolean

Returns:

Either ERR (-1) or OK (0).

1.81. Is_linetouched

Determine if a line has been changed since the last refresh.

Syntax:

Arguments:

n
: Line number.

Returns:

One or zero.

1.82. Is_wintouched

Determine if a window has been changed since the last refresh.

Syntax:

Arguments:

None.

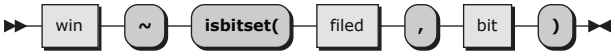
Returns:

One or zero.

1.83. Isbitset

Return true if a bit is set.

Syntax:



Arguments:

field
: Field value to test.

bit
: The bit to test.

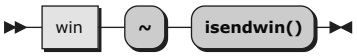
Returns:

One or zero.

1.84. Isendwin

Return true if endwin() has been called.

Syntax:



Arguments:

None.

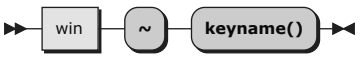
Returns:

One or zero.

1.85. Keyname

Return a string representing the specified input key.

Syntax:



Arguments:

k
: Key value (decimal number).

Returns:

String representing the key name.

1.86. Keypad

Turn on/off reading keypad (function, etc) keys.

Syntax:

Arguments:

bf
: Boolean.

Returns:

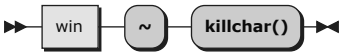
Either ERR (-1) or OK (0).

> Example 1.38. Turn on/off reading keypad (function, etc) keys
>
> ```
> scr = .window~new()
> scr~noecho()
> scr~keypad(.true)
> ```

1.87. Killchar

Return the kill terminal's kill character.

Syntax:



Arguments:

None.

Returns:

String of one character.

1.88. Leaveok

Turn on/off synchrnizing the logical and the hardware cursor location after a refresh.

Syntax:

Arguments:

bf
: Boolean.

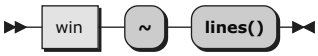
Returns:

Either ERR (-1) or OK (0).

1.89. Lines

Return the number of lines on the stdscr.

Syntax:



Arguments:

None.

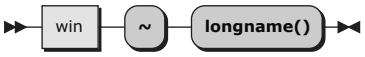
Returns:

Number of lines.

1.90. Longname

Return the terminal string description.

Syntax:



Arguments:

None.

Returns:

String.

1.91. Meta

Turn on/off reading the Alt key info in the 8th bit.

Syntax:

Arguments:

bf
: Boolean.

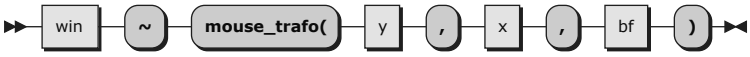
Returns:

Either ERR (-1) or OK (0).

1.92. Mouse_trafo

Translate mouse y and x coordinates to window coordinates.

Syntax:



Arguments:

y
: Y mouse coordinate.

x
: X mouse coordinate.

bf
: Boolean.

Returns:

A string in the form "y x".

1.93. Mousemask

Set/return the mouse event mask.

Syntax:

Arguments:

new
: New mouse mask.

Returns:

Event mask (numeric).

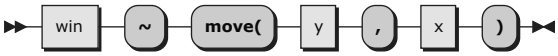
Example 1.39. Set/return the mouse event mask

```
scr = .window~new()

if scr~ncurses_mouse_version() > 0 then do
   scr~addstr('This version of NCurses supports the mouse.' || lf)
   end
else do
   scr~addstr('This version of NCurses does not support the mouse.' || lf)
   end

mmask = scr~mousemask(scr~BUTTON3_CLICKED)
if mmask = scr~BUTTON3_CLICKED then do
   scr~addstr('I am able to read a button 3 click.' || lf)
   end
else do
   scr~addstr('I am not able to read a button 3 click.' || lf)
   end
```

1.94. Move

Move the cursor.

Syntax:



Arguments:

y
: New Y cursor position.

x
: New X cursor position.

Returns:

Either ERR (-1) or OK (0).

Example 1.40. Move the cursor

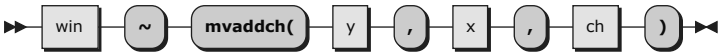
```
scr = .window~new()
scr~noecho()
scr~keypad(.true)
```

```
mmask = scr~mousemask(scr~ALL_MOUSE_EVENTS)

do forever
   ch = scr~getch()
   if ch = scr~KEY_MOUSE then do
      mort = scr~getmouse()
      scr~move(1, 1)
      scr~clrtoeol()
      scr~addstr(mort~y || '/' || mort~x)
      scr~refresh()
      end
   if ch = lf then leave
   end
```

1.95. Mvaddch

Add a single character to the screen after moving the cursor.

Syntax:



Arguments:

y
: Y position.

x
: X position.

ch
: Character to be added. This is a single character, not a numeric value.

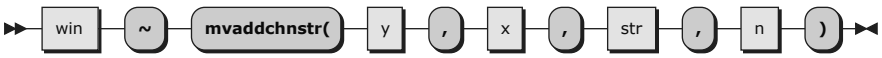
Returns:

Either ERR (-1) or OK (0).

1.96. Mvaddchnstr

Add a chtype character string to the screen after moving the cursor.

Syntax:



Arguments:

y
: Y position.

x
: X position.

str
: Chtype string be added. A chtype is a four-byte sequence.

n
: Number of chtype characters to add.

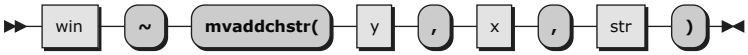
Returns:

Either ERR (-1) or OK (0).

1.97. Mvaddchstr

Add a chtype character string to the screen after moving the cursor.

Syntax:



Arguments:

y
: Y position.

x
: X position.

str
: Chtype string be added. A chtype is a four-byte sequence.

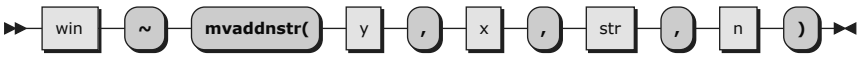
Returns:

Either ERR (-1) or OK (0).

1.98. Mvaddnstr

Add a character string to the screen after moving the cursor.

Syntax:



Arguments:

y
: Y position.

x
: X position.

str
: String to be added.

n
: Number of characters to add.

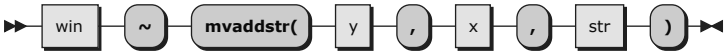
Returns:

Either ERR (-1) or OK (0).

1.99. Mvaddstr

Add a character string to the screen after moving the cursor.

Syntax:



Arguments:

y
: Y position.

x
: X position.

str
: String to be added.

Returns:

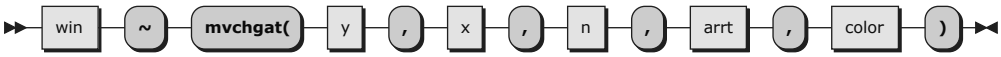
Either ERR (-1) or OK (0).

> Example 1.41. Add a string after moving the cursor
>
> ```
> scr = .window~new()
>
> maxyx = scr~getmaxyx()
> parse var maxyx maxy maxx .
> scr~scrollok(.true)
>
> do y = 1 to maxy
>    scr~mvaddstr(y, 1, "This is boring text written to line" y)
>    end
> ```

1.100. Mvchgat

Change attributes on the window after moving the cursor.

Syntax:



Arguments:

y
: Y start position.

x
: X start position.

n
: Number of character positions.

attr
: The attribute.

color
: The COLOR_PAIR number.

Returns:

Either ERR (-1) or OK (0).

1.101. Mvderwin

Display a different portion of the parent window.

Syntax:

Arguments:

y
: Parent Y position.

x
: Parent X position.

Returns:

Either ERR (-1) or OK (0).

1.102. Mvgetch

Get character from the keyboard after moving the cursor.

Syntax:

Arguments:

y
: New Y position for the cursor.

x
: New X position for the cursor.

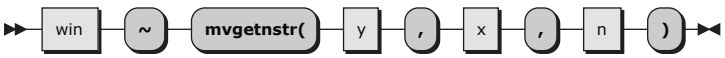
Returns:

A one character string.

1.103. Mvgetnstr

Get a string from the terminal after moving the cursor.

Syntax:



Arguments:

y
: New Y positions for the cursor.

x
: New X positions for the cursor.

n
: Number of characters to get.

Returns:

A character string.

1.104. Mvgetstr

Get a string from the terminal after moving the cursor.

Syntax:

Arguments:

y
: New Y positions for the cursor.

x
: New X positions for the cursor.

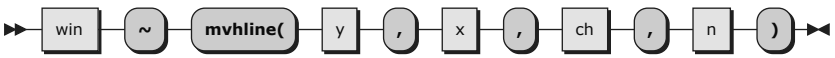
Returns:

A character string (maximum of 1023 characters).

1.105. Mvhline

Draw a horizontal line after moving the cursor.

Syntax:



Arguments:

y
: New Y cursor position

x
: New C cursor position

ch
: Character type (chtype)

n
: Number of characters to draw.

Returns:

Either ERR (-1) or OK (0).

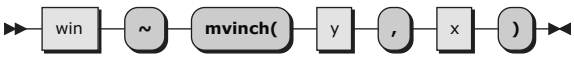
Example 1.42. Draw a horizontal after moving the cursor

```
y = .array~of( 1,  1,  6,  1,  6,  6, 11, 11, 11, 11, 16,  6)
x = .array~of(11, 11,  2, 21, 21, 31,  2, 11, 21, 21, 11,  1)

scr = .window~new()

do c = 1 to 12 by 2
   scr~mvhline(y[c], x[c], 0, 10)
   scr~mvvline(y[c + 1], x[c + 1], 0, 5)
   end
```

1.106. Mvinch

Return the attr/char under the cursor after moving the cursor.

Syntax:



Arguments:

y
: New cursor y position.

x
: New cursor x position.

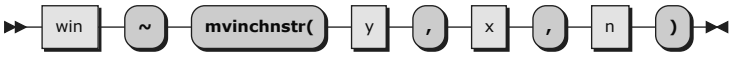
Returns:

A numeric containing the attribut and character.

1.107. Mvinchnstr

Return the attr/char under the cursor after moving the cursor.

Syntax:



Arguments:

y
: New cursor y position.

x
: New cursor x position.

n
: Number of attr/char to return.

Returns:

A chtype array as a character string.

1.108. Mvinchstr

Return the attr/char under the cursor after moving the cursor.

Syntax:

Arguments:

y
: New cursor y position.

x
: New cursor x position.

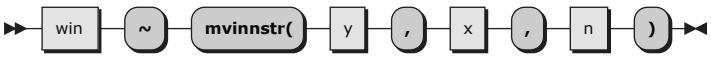
Returns:

A chtype array as a character string (maximum of 1023 bytes).

1.109. Mvinnstr

Read a string from the terminal after moving the cursor.

Syntax:



Arguments:

y
: New y position for the cursor.

x
: New x position for the cursor.

n
: Number of characters to read.

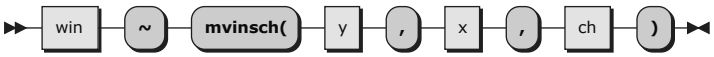
Returns:

A string.

1.110. Mvinsch

Insert the attr/char under the cursor after moving the cursor.

Syntax:



Arguments:

y
: New cursor y position.

x
: New cursor x position.

ch
: chtype to be inserted.

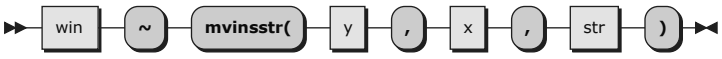
Returns:

Either ERR (-1) or OK (0).

1.111. Mvinsstr

Insert a string after moving the cursor.

Syntax:



Arguments:

y
: New cursor y position.

x
: New cursor x position.

str
: String to insert.

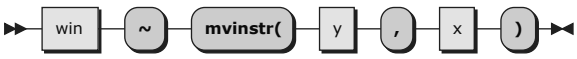
Returns:

Either ERR (-1) or OK (0).

1.112. Mvinstr

Read a string from the terminal after moving the cursor.

Syntax:



Arguments:

y
: New y position for the cursor.

x
: New x position for the cursor.

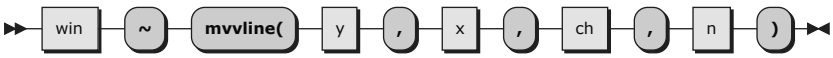
Returns:

A string (maximum of 1023 bytes).

1.113. Mvvline

Draw a vertical line after moving the cursor.

Syntax:



Arguments:

y
: New Y cursor position.

x
: New X cursor position.

ch
: chtype.

n
: Number of rows.

Returns:

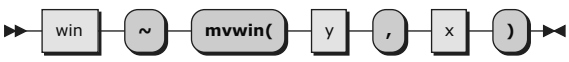
Either ERR (-1) or OK (0).

> Example 1.43. Draw a vertical after moving the cursor
>
> ```
> y = .array~of( 1,  1,  6,  1,  6,  6, 11, 11, 11, 11, 16,  6)
> x = .array~of(11, 11,  2, 21, 21, 31,  2, 11, 21, 21, 11,  1)
>
> scr = .window~new()
>
> do c = 1 to 12 by 2
>    scr~mvhline(y[c], x[c], 0, 10)
>    scr~mvvline(y[c + 1], x[c + 1], 0, 5)
>    end
> ```

1.114. Mvwin

Move a window.

Syntax:



Arguments:

y
: Parent Y position.

x
: Parent X position.

Returns:

Either ERR (-1) or OK (0).

Example 1.44. Move a window

```
tsize = 18
scr = .window~new()

maxyx = scr~getmaxyx()
parse var maxyx maxy maxx .
x = ((maxx - tsize) / 2) + 1
b = .window~new(1, tsize, 1, x)
b~addstr("I'm getting sick!")

do y = 2 to maxy
   b~mvwin(y, x)
   b~refresh()
   scr~touchline(y - 1, 1)
   scr~refresh()
   b~getch()
   end
```

1.115. Napms

Pause the program for number of microseconds.

Syntax:

Arguments:

n
: Number of microseconds

Returns:

Either ERR (-1) or OK (0).

Example 1.45. Pause a program

```
scr = .window~new()

scr~move(1, 1)
scr~addch('*')
scr~refresh()
scr~napms(500)
```

1.116. Ncurses_mouse_version

Return the Ncurses mouse version string.

Syntax:

Arguments:

None.

Returns:

String.

Example 1.46. Return the nCurses mouse version string

```
lf = .window~ASCII_LF~d2c()

scr = .window~new()

if scr~ncurses_mouse_version() > 0 then do
   scr~addstr('This version of NCurses supports the mouse.' || lf)
   end
else do
   scr~addstr('This version of NCurses does not support the mouse.' || lf)
   end
```

1.117. Ncurses_version

Return the Ncurses version string.

Syntax:

Arguments:

None.

Returns:

String.

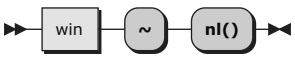
Example 1.47. Return the nCurses version string

```
lf = .window~ASCII_LF~d2c()

scr = .window~new()

if scr~ncurses_mouse_version() > 0 then do
   scr~addstr('This version of NCurses supports the mouse.' || lf)
   end
else do
   scr~addstr('This version of NCurses does not support the mouse.' || lf)
   end
```

1.118. Nl

Distinguish between cr and lf.

Syntax:



Arguments:

None.

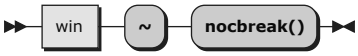
Returns:

Either ERR (-1) or OK (0).

1.119. Nocbreak

Deactivates cbreak (buffering) mode.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

1.120. Nodelay

Transforms getch() function to nonblocking.

Syntax:

Arguments:

bf
: On/off boolean.

Returns:

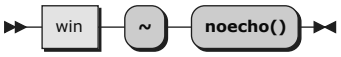
Either ERR (-1) or OK (0).

> Example 1.48. Change the setch() function to non-blocking
>
> ```
> scr = .window~new()
>
> scr~nodelay(.TRUE);  /* turn off getch() wait */
> ```

1.121. Noecho

Turn off echo.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

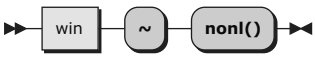
Example 1.49. Turn off echo

```
scr = .window~new()
scr~noecho()
```

1.122. Nonl

Do not distinguish between cr and lf.

Syntax:



Arguments:

None.

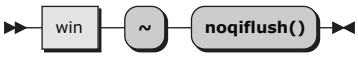
Returns:

Either ERR (-1) or OK (0).

1.123. Noqiflush

Do no flush the input queue on interrupt.

Syntax:



Arguments:

None.

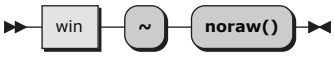
Returns:

Either ERR (-1) or OK (0).

1.124. Noraw

Do not remove modification done by the terminal to input.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

1.125. Notimeout

Pause input function after user presses ESC key.

Syntax:

Arguments:

bf
: On/off boolean.

Returns:

Either ERR (-1) or OK (0).

1.126. Overlay

Overlay the destination window.

Syntax:

Arguments:

dwin
: Destination window.

Returns:

Either ERR (-1) or OK (0).

Example 1.50. Overlay the destination window

```
alpha = .window~new(0,0,1,1)  /* Remember to check for errors! */

scr~addstr(text1)     /* Add text to stdscr and wait */
scr~refresh()
scr~getch()

alpha~addstr(text2)   /* Show win alpha and wait */
alpha~refresh()
alpha~getch()

/* Copy text from one window to the other, non-destructively */
```

```
scr~overlay(alpha)
```

1.127. Overwrite

Overwrite the destination window.

Syntax:

Arguments:

dwin
: Destination window.

Returns:

Either ERR (-1) or OK (0).

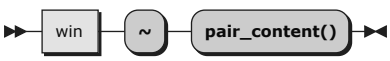
Example 1.51. Overwrite the destination window

```
alpha = .window~new(0,0,1,1)  /* Remember to check for errors! */

scr~addstr(text1)     /* Add text to stdscr and wait */
scr~refresh()
scr~getch()

alpha~addstr(text2)   /* Show win alpha and wait */
alpha~refresh()
alpha~getch()

/* Copy text from one window to the other, destructively */
scr~overwrite(alpha)   -- see book text
```

1.128. Pair_content

Return the pair of colors in a color pair.

Syntax:



Arguments:

num
: Color pair number.

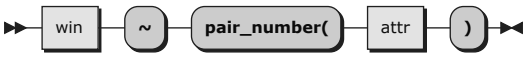
Returns:

String in the form "fg bg" (two numberics).

1.129. Pair_number

Return the default pair attr.

Syntax:



Arguments:

attr
: Color pair attribute.

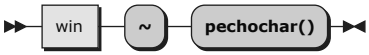
Returns:

Numeric.

1.130. Pechochar

This method is not valid for a window.

Syntax:



Arguments:

None.

Returns:

ERR (-1).

1.131. Pnoutrefresh

This method is not valid for windows.

Syntax:

Arguments:

None.

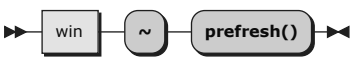
Returns:

ERR (-1).

1.132. Prefresh

This method is not valid for windows.

Syntax:



Arguments:

None.

Returns:

ERR (-1).

1.133. Putwin

Save a new window to a file.

Syntax:

Arguments:

fname
: File name.

Returns:

Either ERR (-1) or OK (0).

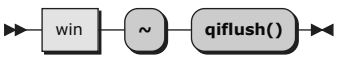
Example 1.52. Save a window to a file

```
scr = .window~new()

retc = win~putwin('window.dat')
if retc = scr~ERR then do
   scr~addstr('Error putting window to disk' || lf)
   end
else do
   scr~addstr('Window put to disk' || lf)
   end
```

1.134. Qiflush

Flush the input queue on interrupt.

Syntax:



Arguments:

Arguments:

None.

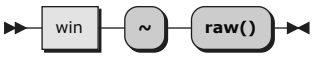
Returns:

Either ERR (-1) or OK (0).

1.135. Raw

Remove modification done by the terminal to input.

Syntax:



Arguments:

None.

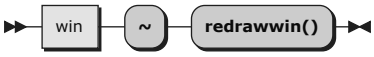
Returns:

Either ERR (-1) or OK (0).

1.136. Redrawwin

Redraw an entire window.

Syntax:



Arguments:

None.

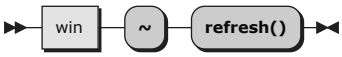
Returns:

Either ERR (-1) or OK (0).

1.137. Refresh

Refresh the screen and turn nCurses formatting on.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

> Example 1.53. Refresh the screen and turn nCurses formatting on

```
scr = .window~new()
scr~addstr('Goodbye, cruel C Programming!')
scr~refresh()
```

1.138. Scr_dump

Dump the terminal screen to a file.

Syntax:

Arguments:

fname
: Output file name.

Returns:

Either ERR (-1) or OK (0).

Example 1.54. Dump the terminal screen to a file

```
scr = .window~new()

retc = scr~scr_dump('windump')
if retc = scr~ERR then do
   scr~addstr('Error writing window to disk' || lf)
   end
else do
   scr~addstr('File written: press Enter to quit' || lf)
   end
```

1.139. Scr_restore

Restore the terminal screen drom a file.

Syntax:

Arguments:

fname
: Input file name.

Returns:

Either ERR (-1) or OK (0).

Example 1.55. Restore the screen from a file

```
scr = .window~new()

retc = scr~scr_restore('windump')
if retc = scr~ERR then do
   scr~addstr('Error reading window file: press Enter' || lf)
   end
```

1.140. Scrl

Scroll a window.

Syntax:

Arguments:

num
: Number of lines to scroll.

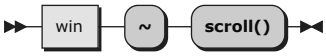
Returns:

Either ERR (-1) or OK (0).

1.141. Scroll

Scroll a window.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

1.142. Scrollok

Allow a window to scroll.

Syntax:

Arguments:

bf
: Boolean indicator.

Returns:

Either ERR (-1) or OK (0).

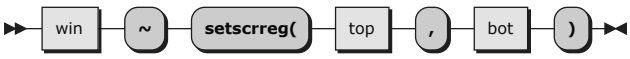
> Example 1.56. Allow a window to scroll

```
scr = .window~new()

scr~scrollok(.true)
```

1.143. Setscrreg

Set a window reagion scrollable.

Syntax:



Arguments:

top
: Top row.

bot
: Bottom row.

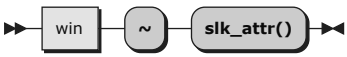
Returns:

Either ERR (-1) or OK (0).

1.144. Slk_attr

Return soft label attr_t.

Syntax:



Arguments:

None.

Returns:

Numeric.

1.145. Slk_attroff

Turn off one or more soft label attributes.

Syntax:

Arguments:

attr
: Attribute(s).

Returns:

Either ERR (-1) or OK (0).

1.146. Slk_attron

Turn on one or more soft label attributes.

Syntax:

Arguments:

attr
: Attribute(s).

Returns:

Either ERR (-1) or OK (0).

1.147. Slk_attrset

Set soft label attributes.

Syntax:

Arguments:

attr
: Attribute(s).

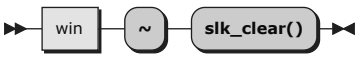
Returns:

Either ERR (-1) or OK (0).

1.148. Slk_clear

Hide the soft labels.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

Example 1.57. Hide the soft labels

```
center = 1
label_text = .array~of('S', 'O', 'F', 'T', 'K', 'E', 'Y', 'S')

.window~slk_init(1)
scr = .window~new()

scr~slk_restore()
do label = 1 to label_text~items()
```

```
   scr~slk_set(label, label_text[label], center)
   end
scr~slk_refresh()
ch = scr~getch()

scr~slk_clear()
ch = scr~getch()
```

1.149. Slk_color

Set soft label colors.

Syntax:

Arguments:

color
: Color pair number.

Returns:

Either ERR (-1) or OK (0).

1.150. Slk_label

Return the soft label text.

Syntax:

Arguments:

num
: Number of the soft label.

Returns:

Label string.

1.151. Slk_noutrefresh

Prepares soft labels for a refresh.

Syntax:

Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

1.152. Slk_refresh

Refresh soft labels.

Syntax:

Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

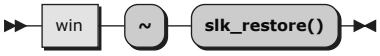
> Example 1.58. Refresh the soft labels

```
left = 0
center = 1
right = 2

.window~slk_init(0)
scr = .window~new()

scr~slk_set(1, 'Help!', left)
scr~slk_set(2, 'File', left)
scr~slk_set(3, 'Print', left)
scr~slk_set(4, 'Text', center)
scr~slk_set(5, 'Edit', center)
scr~slk_set(6, 'Quick', right)
scr~slk_set(7, 'Conf', right)
scr~slk_set(8, 'Change', right)
scr~slk_refresh()
```

1.153. Slk_restore

Restore soft labels.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

> Example 1.59. Restore the soft labels

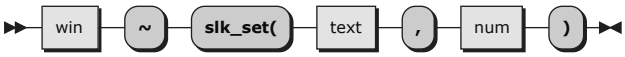
```
.window~slk_init(3)
```

```
scr = .window~new()

scr~slk_restore()
```

1.154. Slk_set

Set soft label text.

Syntax:



Arguments:

text
: Soft label text.

num
: Soft label number.

Returns:

Either ERR (-1) or OK (0).

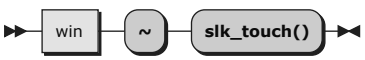
Example 1.60. Set the soft labels

```
left = 0
center = 1
right = 2

.window~slk_init(0)
scr = .window~new()

scr~slk_set(1, 'Help!', left)
scr~slk_set(2, 'File', left)
scr~slk_set(3, 'Print', left)
scr~slk_set(4, 'Text', center)
scr~slk_set(5, 'Edit', center)
scr~slk_set(6, 'Quick', right)
scr~slk_set(7, 'Conf', right)
scr~slk_set(8, 'Change', right)
scr~slk_refresh()
```

1.155. Slk_touch

Touch soft labels.

Syntax:



Arguments:

None.

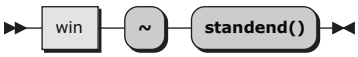
Returns:

Either ERR (-1) or OK (0).

1.156. Standend

Turn off text attributes.

Syntax:



Arguments:

None.

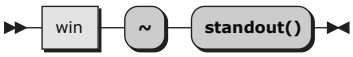
Returns:

Either ERR (-1) or OK (0).

1.157. Standout

Turn on text attributes.

Syntax:



Arguments:

None.

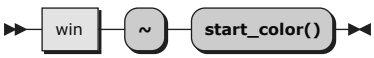
Returns:

Either ERR (-1) or OK (0).

1.158. Start_color

Turn on using color attributes.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

> Example 1.61. Turn on using color

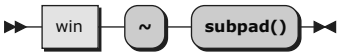
```
scr = .window~new()
```

```
scr~start_color()
```

1.159. Subpad

This method is not valid for a window.

Syntax:



Arguments:

None.

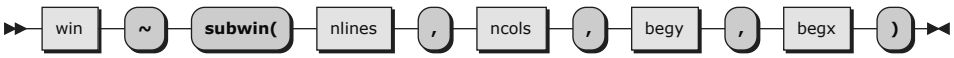
Returns:

ERR (-1).

1.160. Subwin

Create a subwindow.

Syntax:



Arguments:

nlines
: Number of lines.

ncols
: Number of cols.

begy
: Beginning Y line.

begx
: Beginning X column.

Returns:

A Window instance.

Example 1.62. Create a subwindow

```
scr = .window~new()

sonny = scr~subwin(5, 20, 11, 31)
```

1.161. Syncok

Turn on/off aut touch for a window.

Syntax:

Arguments:

bf
: On/off boolean.

Returns:

Either ERR (-1) or OK (0).

1.162. Tabsize

Return/set the tab size.

Syntax:

Arguments:

sz
: New tab size.

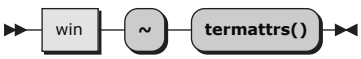
Returns:

Tab size (numeric).

1.163. Termattrs

Return the which attributes the terminal is capable of producing.

Syntax:



Arguments:

None.

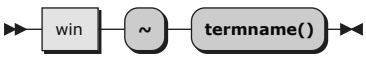
Returns:

Numeric.

1.164. Termname

Return the terminal name.

Syntax:



Arguments:

None.

Returns:

String.

1.165. Timeout

Set the timeout for text input.

Syntax:

Arguments:

tmout
: Timeout value.

Returns:

Either ERR (-1) or OK (0).

1.166. Touchline

Mark one or more lines as changed.

Syntax:

Arguments:

bf
: On/off boolean.

Returns:

Either ERR (-1) or OK (0).

> Example 1.63. Mark one or more lines as changed

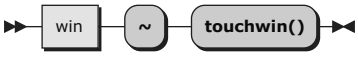
```
tsize = 18
scr = .window~new()

maxyx = scr~getmaxyx()
parse var maxyx maxy maxx .
x = ((maxx - tsize) / 2) + 1
b = .window~new(1, tsize, 1, x)
b~addstr("I'm getting sick!")

do y = 2 to maxy
   b~mvwin(y, x)
   b~refresh()
```

```
    scr~touchline(y - 1, 1)
    scr~refresh()
    b~getch()
    end
```

1.167. Touchwin

Mark window as changed.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

1.168. Typeahead

Set the typeahead for text input.

Syntax:

Arguments:

fd
: Typeahead value.

Returns:

Either ERR (-1) or OK (0).

1.169. Unctrl

Convert a control code to a ^code sequence.

Syntax:

Arguments:

ch
: Code to convert.

Returns:

Either ERR (-1) or OK (0).

1.170. Ungetch

Put a character into the keyboard buffer.

Syntax:

Arguments:

ch
: Character to insert.

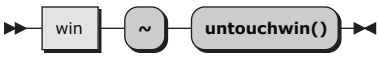
Returns:

Either ERR (-1) or OK (0).

1.171. Untouchwin

Unmark window as changed.

Syntax:



Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

1.172. Use_default_colors

Use the default terminal colors.

Syntax:

Arguments:

None.

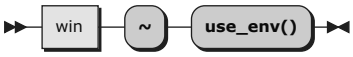
Returns:

Either ERR (-1) or OK (0).

1.173. Use_env

Use the default terminal colors.

Syntax:



Arguments:

None.

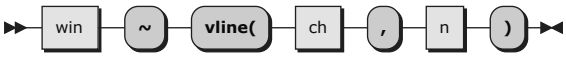
Returns:

Either ERR (-1) or OK (0).

1.174. Vline

Draw a vertical line.

Syntax:



Arguments:

ch
: chtype.

n
: Number of rows.

Returns:

Either ERR (-1) or OK (0).

Example 1.64. Draw a vertical line

```
scr = .window~new()

maxyx = scr~getmaxyx()
parse var maxyx maxy maxx .
x = 1
do y = 1 to maxy
   scr~move(y, x)
   scr~hline(0, maxx - x + 1)
   scr~vline(0, maxy - y + 1)
   x += 2
   end
```

1.175. Wenclose

Determine if a mouse click are within a window (self).

Syntax:

Arguments:

y
: Y cursor screen position.

x
: X cursor screen position.

Returns:

One or zero.

1.176. Wnoutrefresh

Copy modified text to the screen.

Syntax:

Arguments:

None.

Returns:

Either ERR (-1) or OK (0).

1.177. Wredrawln

Redraw a line to the screen.

Syntax:

Arguments:

beg
: Beginning line number.

n
: Number of lines.

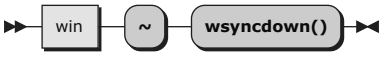
Returns:

Either ERR (-1) or OK (0).

1.178. Wsyncdown

Ensure subwindows are touched when the main window is touched.

Syntax:



Arguments:

None.

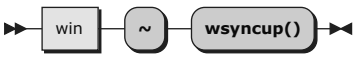
Returns:

Either ERR (-1) or OK (0).

1.179. Wsyncup

Ensure parent windows are touched when the sub window is touched.

Syntax:



Arguments:

None.

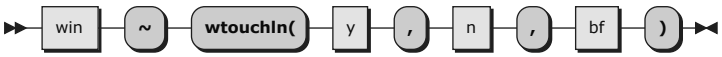
Returns:

Either ERR (-1) or OK (0).

1.180. Wtouchln

Mark one or more lines as changed.

Syntax:



Arguments:

y
: Start line number.

n
: Number of lines.

bf
: Changed/Unchanged boolean.

Returns:

Either ERR (-1) or OK (0).

Pad Class Method Reference

2.1. New - Class Method

Create a new pad.

Syntax:

Arguments:

nlines
: Number of lines.

ncols
: Number of columns.

Returns:

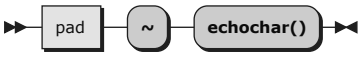
A new pad instance.

> Example 2.1. Create a new pad
>
> ```
> /* create a new pad */
> win = .Window~new(5, 10)
> ```

2.2. Echochar

This method is not valid foa a pad.

Syntax:



Arguments:

None.

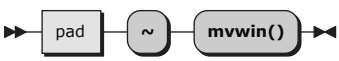
Returns:

ERR (-1).

2.3. Mvwin

This method is not valid for a pad.

Syntax:



Arguments:

None.

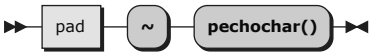
Returns:

ERR (-1).

2.4. Pechochar

Echo one character to a pad and the stdscr..

Syntax:



Arguments:

ch
: Character to echo.

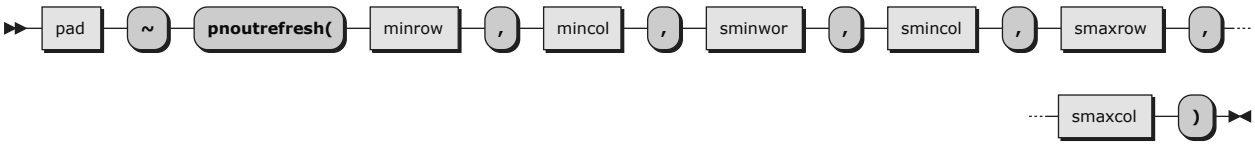
Returns:

Either ERR (-1) or OK (0).

2.5. Pnoutrefresh

Refresh a pad.

Syntax:



Arguments:

minrow
: Minimum row number.

mincol
: Minimum col number.

sminrow
: Minimum row number.

smincol
: Minimum col number.

smaxrow
: Maximum row number.

smaxcol
: Maximum col number.

Returns:

Either ERR (-1) or OK (0).

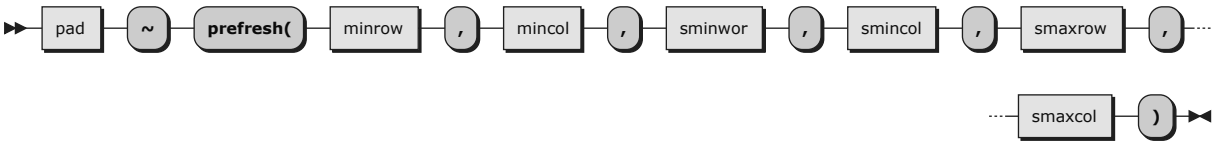
Example 2.2. Refresh a pad

```
filename = 'Readme.txt'
tall = 24
wide = 19
spacer = 5

scr = .window~new()
p = .pad~new(200, wide + 1)
s1 = p~subpad(tall, wide + 1, 1, 1)
s2 = p~subpad(tall, wide + 1, tall + 1, 1)
s3 = p~subpad(tall, wide + 1, (2 * tall) + 1, 1)

strm = .stream~new(filename)
strm~open('read')
eof = .false
call on notready name eof

ch = strm~charin()
do while \eof
   p~addch(ch)
   ch = strm~charin()
   end
strm~close()

s1~pnoutrefresh(1, 1, 1, 1, tall, wide + 1)
s2~pnoutrefresh(1, 1, 1, wide + spacer + 1, tall, (wide * 2) + spacer)
s3~pnoutrefresh(1, 1, 1, (wide * 2) + (spacer * 2), tall, (wide *3) + (spacer * 2))
scr~doupdate()
```

2.6. Prefresh

Refresh the pad to the stdscr.

Syntax:



Arguments:

minrow
: Minimum row number.

mincol
: Minimum col number.

sminrow
: Minimum row number.

smincol
: Minimum col number.

smaxrow
: Maximum row number.

smaxcol
: Maximum col number.

Returns:

Either ERR (-1) or OK (0).

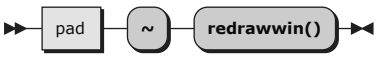
Example 2.3. Refresh the pad to a stdscr

```
scr = .window~new()

p = .pad~new(50, 100)
if p = .nil then do
   scr~addstr("Unable to create pad")
   scr~refresh()
   scr~getch()
   scr~endwin()
   return
   end

scr~addstr("New pad created")
scr~refresh()
p~addstr("New pad created")
p~prefresh(1, 1, 1, 1, 2, 16)
```

2.7. Redrawwin

This method is not valid for a pad.

Syntax:



Arguments:

None.

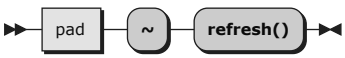
Returns:

ERR (-1).

2.8. Refresh

This method is not valid for a pad.

Syntax:



Arguments:

None.

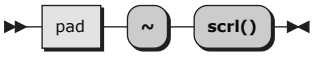
Returns:

ERR (-1).

2.9. Scrl

This method is not valid for a pad.

Syntax:



Arguments:

None.

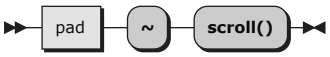
Returns:

ERR (-1).

2.10. Scroll

This method is not valid for a pad.

Syntax:



Arguments:

None.

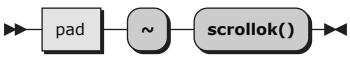
Returns:

ERR (-1).

2.11. Scrollok

This method is not valid for a pad.

Syntax:



Arguments:

None.

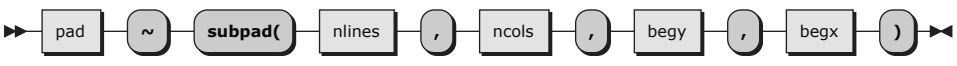
Returns:

ERR (-1).

2.12. Subpad

Create a subpad.

Syntax:



Arguments:

nlines
: Number of lines.

ncols
: Number of cols.

begy
: Beginning Y line.

begx
: Beginning X column.

Returns:

A Pad instance.

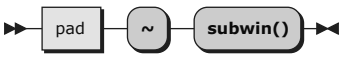
Example 2.4. Create a subpad

```
lf = .window~ASCII_LF~d2c()

scr = .window~new()

pod = .pad~new(50, 50)

scr~addstr("New pad created" || lf)
scr~refresh()

pea = pod~subpad(20, 20, 30, 30)
```

2.13. Subwin

This method is not valid for a pad.

Syntax:



Arguments:

None.

Returns:

ERR (-1).

2.14. Wnoutrefresh

This method is not valid for a pad.

Syntax:

Arguments:

None.

Returns:

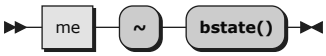
ERR (-1).

Mevent Class Method Reference

3.1. Bstate

The mouse button state.

Syntax:



Arguments:

None.

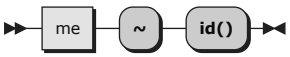
Returns:

Mouse button state (numeric).

3.2. Id

The mouse event pointing device ID.

Syntax:



Arguments:

None.

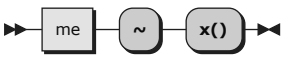
Returns:

Mouse event ID (numeric).

3.3. X

The mouse column number position.

Syntax:



Arguments:

None.

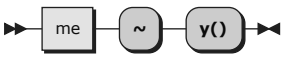
Returns:

Mouse column position (numeric).

3.4. Y

The mouse row number position.

Syntax:



Arguments:

None.

Returns:

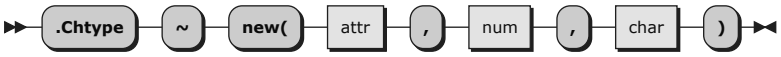
Mouse row position (numeric).

Chtype Class Method Reference

4.1. New

Create a chtype instance.

Syntax:



Arguments:

attr
: The attribute (numeric).

num
: A color pair number (numeric).

char
: A single character.

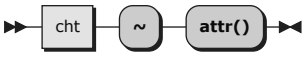
Returns:

A new Chtype instance.

4.2. Attr

Returns/sets the numeric attribute value of the chtype.

Syntax:



Arguments:

None.

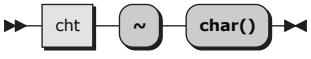
Returns:

The numeric chtype attribute value (numeric).

4.3. Char

Returns/sets the character of the chtype.

Syntax:



Arguments:

None.

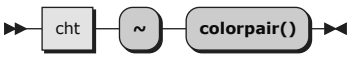
Returns:

The chtype character value.

4.4. Colorpair

Returns/sets the numeric color pair value of the chtype.

Syntax:



Arguments:

None.

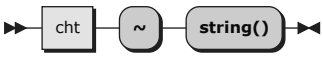
Returns:

The numeric chtype color pair value (numeric).

4.5. String

Returns the numeric value of the chtype.

Syntax:



Arguments:

None.

Returns:

A numeric chtype value (numeric).

Appendix A. Sample nCurses Programs

The OrxnCurses distribution contains a number of example programs. All of these progeams are ported from example programs contained in the book *Programmer's Guide to nCurses* by Dan Gookin published by Wiley Technology Publishing® ISBN: 978-0-470-10759-1.

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.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

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 Aug 2016

Initial creation for 5.0

Index

T

U

V

W

X

Y