BREXX/370 V2R5M2 Formatted Screens

Document Version V2R5M2
Authors: Peter Jacob (pej), Mike Großmann(mig)

The following document is a brief description of the new Formatted Screen (FSS) feature. It allows the setup of simple screen definitions within a BREXX script.

For detail take a closer look at the FSS samples in the delivered Installation library
BREXX.INSTALL.SAMPLES

1 Delivered Samples

The relevant FSS samples are prefixed with the #-sign:

#TSOAPPL	Shows in a detailed usage of all FSS functions how to set up a menu and “paint” a TK4 like design
#BROWSE	A pre-packed FSS application to display data in a List Buffer instead of using SAYs
#FSS1COL	A pre-packed FSS application to generate input requests (in one column)
#FSS2COL	A pre-packed FSS application to generate input requests (distributes in two columns)
#FSS3COL	A pre-packed FSS application to generate input requests (distributes in three columns)
#FSS4COL	A pre-packed FSS application to generate input requests (distributes in four columns)
#FSS4CLX	A pre-packed FSS application to generate input requests (distributes in four columns) With additional setting options, including all callback to test user’s input

2 FSS Limitation

The FSS screen limitation has been dropped. Now large screen widths and heights are supported.

FSS supports just one FSS Screen definition at a time. If you need to display more than one FSS Screen in your REXX application, you must close the first and set up and display the next FSS definition. Using this method, you can easily switch between different FSS Screens. It is a good idea to separate the FSS definitions into different sub-procedures; this allows their display by calling it.

3 FSS Function Overview

To use FSS functions in BREXX, you must import the FSS API library from BREXX.RXLIB, address and initialise it by a call to FSSINIT, be aware that FSS is a host command application that requires an ADRESS FSS command, it is sufficient to use it once at the beginning. From this time on all host, commands are directed to FSS. If it happens to be and you have to switch to another host API (e.g. ADDRESS TSO or ADDRESS SYSTEM), you can do so, but you must make sure to switch back to the FSS API by re-issuing an ADDRESS FSS command:

```
/* IMPORT THE API LIBRARY */
CALL IMPORT FSSAPI
/* ADDRESS THE FSS SUBSYSTEM */
ADDRESS FSS
/* SWITCH TO FULL-SCREEN MODE */
CALL FSSINIT
```

3.1 FSSINIT Inits the FSS subsystem

Initialise the FSS environment; this must be performed before any other FSS call.

```
CALL FSSINIT
```

3.2 Principles of Defining Formatted Screens

You can define your formatted screen by using a series of FSSTEXT and FSSFIELD and/or some wrapped FSS functions as FSSMESSAGE, FSSCOMMAND, etc. in your REXX script. Essential parameters are, in all cases, the ROW and COLUMN positions. Be aware that consistency validations are very basic and not bulletproof at all. It is, for example, possible to accidentally re-use occupied ranges, which may lead to unwanted behaviour or results. Performing just necessary validations increases the performance of the screen handling. It is, therefore, essential that you carefully design your Formatted Screens.

3.3 FSSTEXT Display a text field

```
CALL FSSTEXT 'text' ,row,column,[text-length],attributes
```

text: text to be displayed in the screen
row: row where text should be placed
column: column where text should be placed.

text-length: length occupied by the text, this is an optional parameter; it defaults to the text length.
attributes: screen attributes, like colours, protected, high-lighted etc. For details refer to the attributes section

3.4 FSSFIELD Display an input field and associate it with a BREXX Variable

```
CALL FSSFIELD 'field',row,column,[length],attributes[,init-value]
```

field: field-name of an input area to be displayed on the screen
row: row where text should be placed
column: column where the input area should be placed
length: the length occupied by the text, this is an optional parameter, it defaults to the text length.
attributes: screen attributes, like colours, protected, high-lighted etc. For details refer to the attributes section
init-value what should be displayed as content of the input field. It defaults to blank.

3.4.1 Important Notice on the Column Position

Each text or field definition starts with the defined attribute byte, which itself is invisible but tells how the text or field appears on the screen. Therefore the original text or field-definition start at column+1.

3.4.2 Important Notice on Screen Definitions

Be aware that all definitions provided by FSSTEXT and FSSFIELD are stacked internally. They do not create a formatted screen on the fly.

This can be achieved by calling **CALL FSSDISPLAY** (documented separately in this document)

3.4.3 Attribute Definition

The attribute definitions trigger the behaviour or colours of the Formatted Screen text or input elements.

#PROT	Definition is protected (default for fsstext)
#NUM	input field must be numeric
#HI	text is displayed high-lighted
#NON	text/field-input is invisible
#BLINK	text/field blinks
#REVERSE	background is set with defined colour text appears white
#USCORE	Underscore field

Colours:

#BLUE	text or input field is of blue colour
#RED	text or input field is of red colour
#PINK	text or input field is of pink colour
#GREEN	text or input field is of green colour
#TURQ	text or input field is of turquoise colour
#YELLOW	text or input field is of yellow colour
#WHITE	text or input field is of white colour

You can combine several attribute bytes by adding them.
e.g. #PROT+#BLUE
combining several colours is not allowed and may lead to unexpected errors

3.5 FSSTITLE Displays a centred Title in Screen line 1

```
CALL FSSTITLE 'title-text[,attributes]
```

Besides the title definition the right hand 25 bytes may contain a short message in case of errors, it overwrites the title part in error situations and automatically resets it, if the enter key is used.

The error field is named **ZERRSM** and maybe set also by your program.

3.6 FSSOPTION Create OPTION Line

Creates an OPTIONs line, typically used in a menu to select a menu option.

```
OPTION ===> _____________________________________________________
```

```
CALL FSSOPTION [row[,option-length[,attribute1,[attribute2]]]
```

row	defaults to 2
option-length	defines the line length to proved the option input, default is length of the remaining line
attribute1	Attribute of “OPTION”, default is #PROT+#WHITE
attribute2	Attribute of the option line,default is #HI+#RED+#USCORE

3.7 FSSCOMMAND Create a Command Line

Creates an input line for entering menu options or commands, it appears with the “COMMAND ===>” prefix and is typically located in row 2.

```
COMMAND ===> _____________________________________________________
```

CALL FSSCOMMAND [row[,option-length[,attribute1,[attribute2]]]

row	defaults to 2
option-length	defines the line length to provide the command input, default is length of the remaining line
attribute1	Attribute of “COMMAND”, default is #PROT+#WHITE
attribute2	Attribute of the command line,default is #HI+#RED+#USCORE

3.8 FSSTOPLINE Create an Option/Command Line

FSSTOPLINE is a variation of FSSCOMMAND which allows the free definition of the input line prefix. It is typically located in row 2.

```
MY-OPTION ===> _____________________________________________________
```

CALL FSSTOPLINE prefix,[row[,option-length[,attribute1,[attribute2]]]

Prefix	String which should appear in front of the input line. In the example above it is “MY-OPTION”
row	defaults to 2
option-length	defines the line length to provide the command input; default is the length of the remaining line
attribute1	Attribute of “COMMAND”, default is #PROT+#WHITE
attribute2	Attribute of the command line, default is #HI+#RED+#USCORE

3.9 FSSMESSAGE Create a Message Line

Creates a message line to display messages. The message line occupies a full-screen line.

CALL FSSMESSAGE [row[,attribute]]

row	defaults to 3
attribute	attribute of message line, default is #PROT+#HI+#RED

A call to FSSZERRLM sets the Message

3.10 FSSZERRSM Set Error/Warning/Info Short Message

The message is set in Field ZERRSM. ZERRSM is automatically created by using an FSSTITLE definition; otherwise, it must be defined explicitly. If implicitly used with the FSSTITLE definitions, it starts on the right-hand side after the end of the message; its length is dependant on the length of the title.

```
CALL FSSZERRSM 'message'
```

3.11 FSSZERRLM Set Error/Warning/Info Long Message

The message is set in Field ZERRLM, which has been defined on the screen by a CALL FSSMESSAGE.

```
CALL FSSZERRLM 'message'
```

3.12 FSSFSET Set Field Content

```
CALL FSSFSET 'field',content
```

Make sure the field-name is enclosed in quotes; otherwise, there is a chance of unwanted substitution by its value!

3.13 FSSFGET Get current Field Content

```
Value=FSSFGET('field')
```

Make sure the field-name is enclosed in quotes; otherwise, there is a chance of unwanted substitution by its value!

3.14 FSSFGETALL Get Contents of all Fields

```
Number=FSSFGETALL()
```

All field contents of the screen are fetched and stored in the associated BREXX fields (defined by FSSFIELD(…)

3.15 FSSCURSOR Set Cursor to a Field

```
CALL FSSCURSOR 'field'
```

3.16 FSSCOLOUR Change Colour of a Field

```
CALL FSSCOLOUR 'field',colour-attribute alternatively
CALL FSSCOLOR 'field' ,colour-attribute
```

3.17 FSSKEY Return Key entered

When the user presses an action-key on a screen the used key value to return control can be accessed by FSSKEY. The optional parameter CHAR returns it in a translated readable form if not set the value returned is the decimal value assigned to the action key.

```
key=FSSKEY([CHAR])
```

By FSS supported keys:

REXX Variable	Numeric value	Translated value
#ENTER	125	ENTER
#PFK01	241	PF01
#PFK02	242	PF02
#PFK03	243	PF03
#PFK04	244	PF03
#PFK05	245	PF05
#PFK06	246	PF06
#PFK07	247	PF07
#PFK08	248	PF08
#PFK09	249	PF09
#PFK10	122	PF10
#PFK11	123	PF11
#PFK12	124	PF12
#PFK13	193	PF13
#PFK14	194	PF14
#PFK15	195	PF15
#PFK16	196	PF16
#PFK17	197	PF17
#PFK18	198	PF18
#PFK19	199	PF19
#PFK20	200	PF20
#PFK21	201	PF21
#PFK22	74	PF22
#PFK23	75	PF23
#PFK24	76	PF24
#CLEAR	109	CLEAR
#RESHOW	110	RESHOW

3.18 FSSDISPLAY Display/Refresh a generated Formatted Screen

Displays or Re-Displays the active screen

```
CALL FSSDISPLAY or

CALL FSSREFRESH
```

3.19 Get Screen Dimensions

```
width=FSSWidth()
```
returns the number of available columns defined by the Emulation

height=FSSHeight() returns the number of available rows defined by the Emulation

3.20 Close FSS Environment

Once the Screen Handling is finished it is recommended to terminate the FSS environment

CALL FSSTERM or

CALL FSSTERMINATE or

CALL FSSCLOSE

4 Creating a Dialog Manager

To handle user's action-keys, you can set up a simple Dialog Manager, as shown in this example:

```
/* ------------------------------------------------------------------
 * Display screen in primitive Dialog Manager and handle User's Input
 * ------------------------------------------------------------------
 */
  do forever
     fsreturn=fssDisplay()            /* Display Screen   */
     if fsreturn='PFK03' then leave   /* QUIT requested   */
     if fsreturn='PFK04' then leave   /* CANCEL requested */
     if fsreturn='PFK15' then leave   /* QUIT requested   */
     if fsreturn='PFK16' then leave   /* CANCEL requested */
     if fsreturn<>'ENTER' then iterate
     call fSSgetD()                    /* Read Input Data */
  /* Add input checking if needed */
  end
  call fssclose                       /* Terminate Screen Environment */
```

5 Simple Screen Applications

There is a simple way to create formatted screens using preformatted rexx scripts, and this allows an easy screen setup without coding all the screen definitions manually.

5.1 Screen with Attributes in one Column

```
/*          + ------------------- Screen with 1 column
  *          !
  *          !    + -------------- Title line of screen
  *          !    !     */
frc=FMTCOLUM(1,'One Columned Formatted Screen',
     ,'1. First Name  ===>',
     ,'2. Family Name ===>',
     ,'3. UserId      ===>',
     ,'4. Department  ===>',
     )
do i=1 to _screen.input.0
   say "User's Input "i'. Input Field: '_screen.input.i
end
return
```

The above definition creates and displays this screen:

```
----------------------- One Columned Formatted Screen --------------------

 1. First Name  ===> _____________________________________________________
 2. Family Name ===> _____________________________________________________
 3. UserId      ===> _____________________________________________________
 4. Department  ===> _____________________________________________________
```

After entering input and pressing enter, you receive the provided input

```
----------------------- One Columned Formatted Screen --------------------

 1. First Name  ===> Fred_____________________________________________
 1. Family Name ===> Flintstone_______________________________________
 2. UserId      ===> FL2311___________________________________________
 3. Department  ===> Quarry___________________________________________
```

The provided input is stored in SCREEN.INPUT.xx an can be used or printed as in this REXX script:

```
User's Input 1. Input Field: Fred_____________________________________
User's Input 2. Input Field: Flintstone_______________________________
User's Input 3. Input Field: FL2311___________________________________
User's Input 4. Input Field: Quarry___________________________________
```

5.2 Screen with Attributes in two Columns

By changing the column numbers to 2:

```
/*           + ------------------- Screen with 2 columns
  *          !
  *          !     + -------------- Title line of screen
  *          !     !      */
frc=FMTCOLUM(2,'Two Columned Formatted Screen',
     ,'1. First Name  ===>',
     ,'2. Family Name ===>',
     ,'3. UserId      ===>',
     ,'4. Department  ===>',
     )
do i=1 to _screen.input.0
   say "User's Input "i'. Input Field: '_screen.input.i
end
return
```

you get the attributes in two columns

```
------------------------ Two Columned Formatted Screen ---------------------

1. First Name  ===> ____________________ 2. Family Name ===> _________________
3. UserId      ===> ____________________ 4. Department  ===> _________________
```

Entered input is provided in the same way as in the one column screen example.

5.3 Screen with Attributes in three Columns

```
------------------------ Three Columned Formatted Screen -------------------

 1. First Name  ===> _____ 2. Family Name ===> _____ 3. UserId      ===> ___
 4. Department  ===> _____
```

Just change the number of columns to 3

```
frc=FMTCOLUM(3,'Three Columned Formatted Screen',
…
```

5.4 Screen with Attributes in four Columns

Last option is to place the attributes in four columns:

```
frc=FMTCOLUM(4,'Four Columned Formatted Screen',
…
```

5.5 Screen special Attributes

You can tailor the appearance of formatted column screens, by setting **_screen.xxxx** variables:

5.5.1 Presetting Screen input fields

Use **_SCREEN.INIT.n**='input-value-as-default', n is the reference to the field in the FMTCOLUMN definition. 1 is first, 2 second, etc.

Example:

```
_SCREEN.INIT.1='FRED'
_SCREEN.INIT.3='Flintstone'
_SCREEN.INIT.4='FL2311'
_SCREEN.INIT.5='Quarry'
```

Calling the formatted screen, you get a pre-set Screen:

```
----------------------- One Columned Formatted Screen --------------------

 1. First Name  ===> Fred_____________________________________________
 1. Family Name ===> Flintstone_______________________________________
 2. UserId      ===> FL2311___________________________________________
 3. Department  ===> Quarry___________________________________________
```

5.5.2 Input field appearance

If not changed, the input fields appear with an underscore in the available length. You can change it by setting _screen.preset. If you set **_screen.preset='+'** (one character) the input field filled by the character you defined. If you use more than one character **_screen.preset='_ '** only the given string is displayed.

5.5.3 Input field length

The field length is, by default, delimited by the following field definition in the row, or by the end of the line.

If you want to limit it to a certain length by:

_SCREEN.LENGTH.n=field-length

n is the field number you want to set. It is sufficient to set just the field length you want to limit.

5.5.4 Input Field CallBack Function

Normally, if you press enter, the screen control is giving back to your rexx, and the variable content is returned. If you prefer to check the entered input while your formatted screen is still active, for example, to validate user's input, you can define a callback function:

_screen.ActionKey='internal-subprocedure'

The internal sub-procedure must be coded without a PROCEDURE statement; else you cannot use the screen input variables

```
_screen.ActionKey='checkInput'
frc=FMTCOLUM(2,'Two Columned Formatted Screen',
…
return
/* ---------------------------------------------------------------
  * Call Back Routine from FMTCOLUMN to check provided Input
```

```
  * ---------------------------------------------------------------------
  */
 checkInput:
  if _screen.input.1 = '' then do
     call FSSzerrsm 'Field 1 ist mandatory'
     call FSSzerrlm 'Please enter valid content in Field 1'
     return 1
  end
  if _screen.input.2 = '' then do
     call FSSzerrsm 'Field 2 is mandatory'
     call FSSzerrlm 'Please enter valid content in Field 2'
     return 1
  end
…
```

In case of an error, your call back function can use the **FSSzerrsm** function, which displays a short message in the formatted screen's title line and/or the **FSSzerrlm** function to display a long message. The error message is displayed in the last line of Formatted Screen.

Your callback sub-procedure signals with its return code how to proceed:

return 0	everything ok, leave screen an pass control back to calling rexx
return 128	something is wrong, re-display the screen
return 256	something is wrong, leave the screen
return n:	field n contains wrong input, re-display screen n >0 and n<128 represents the field number in error

5.6 FSSMENU Supporting Menu Screens

5.6.1 FSSMENU Defining a Menu Screen

To ease the creation of menu screens, you can use the FSSMENU definition. It creates the screen layout as well as the dialogue handling part.

```
CALL FSSMENU 'option','note','description','action',[startRow],[startCol]
```

option option code which leads to performing the associated action. The option can be a numeric or alphanumeric string and its length must not exceed 2.

note short description of the action to perform

description long description of the action to perform

action action is performed is associated option is selected
TSO prefixes an action for a TSO function call
or with CALL if a REXX procedure should be called.

startRow row in which the first menu should be placed, **default is 12**. This parameter is only validated for the first FSSMENU definition and automatically used for each subsequent call.
To achieve a row centred menu appearance, you can use the following rexx coding before the first FSSMENU definition:

```
menumax=5 /* number of Menu entries
startRow=(FSSHeight()%2)-(menuMax%2+1)-3
```

and pass startRow as a parameter in the FSSMENU definition

startCol column in which the menu should be placed, **default is 6**. This parameter is only validated for the first FSSMENU definition and automatically used for each subsequent call.
To achieve a column centred menu appearance, you can use the following rexx coding before the first FSSMENU definition:

```
startcol=(FSSWidth()%2)-30
```

and pass startCol as a parameter in the FSSMENU definition

The FSS menu definitions can be included within a typical FSS Screen definition to add additional fields or text parts to the formatted screen. These parts can be dynamically updated if you specify a callback procedure in the FSSMENU Display call.
The FSSMENU definition relies on the existence of the following fields (FSSMENU does not automatically generate them); they must be defined separately, either implicitly or explicitly:

ZCMD is defined by FSSTOPLINE or FSSCOMMAND

ZERRSM **is** defined by FSSTITLE

Example defined in a REXX script:

```
…
 CALL FSSMENU 1,"RFE",    'SPF like" productivity tool',
               ,"TSO CALL 'SYS2.CMDLIB(RFE)"
 CALL FSSMENU 2,"RPF"    ,'SPF like" productivity tool','TSO RPF'
 CALL FSSMENU 3,"IM"     ,'IMON/370 system monitor','TSO IM'
 CALL FSSMENU 4,"QUEUE"  ,'spool browser',          'TSO Q'
 CALL FSSMENU 5,"HELP"   ,'general TSO help',       'TSO HELP'
 CALL FSSMENU 6,"UTILS"  ,
     ,'information on utilities and commands available','TSO HELP UTILS'
```

```
 CALL FSSMENU 7,"TERMTEST" ,'verify 3270 terminal capabilities',
              ,'TSO TERMTEST'
…
```

5.6.2 FSSMENU Displaying a Menu Screen

To display the menu and handle the selected actions, FSSMENU must be called with the $DISPLAY parameter:

returnkey=FSSMENU('$DISPLAY',<callback-procedure>,<actionkey-procedure>)

returnkey key used to end the dialogue handling, it is either PF03, PF04, PF15, or PF16

$DISPLAY Display the menu defined before

callback-procedure optional own callback procedure (internal or external) to update FSS variables or other variables. This procedure is called just before the menu is displayed and re-displayed. Therefore the variables which are defined for the menu screen and modified in the procedure are displayed with their new content.
The callback procedure needs the scope of the FSSMENU variables; therefore, it **must not be defined** with a **PROCEDURE** statement. Just define the callback name with a label.

actionkey-procedure optional own action key procedure (internal or external) to check user's input in the command line. This procedure is called when the user pressed the enter key, and the command line contains input. This input could be a simple menu option or maybe a command, which you like to process. It is also called if a PF-Key was used to request an action. **PF03, PF04, PF15** and **PF16** are not passed to the procedure as they trigger the standard return action
The action key procedure is called with the parameters action-key and command-line. To receive them in your procedure use:

parse arg action, command

Name of the above variables is of course freely selectable
To return to the calling menu, it is essential to provide a return code; this allows the menu processing to decide on the next steps.
Return codes:

- **0** input has been handled by the exit, re-display Menu
- **4** input has not been handled, continue with internal checks
- **8** exit Menu immediately

Example: Simple Display without any exits

```
rckey=FSSMENU('$DISPLAY')
say 'End Key 'rckey
…
```

Example: Before Display update some variables via a callback procedure

```
rckey=FSSMENU('$DISPLAY','UPDVAR')
say 'End Key 'rckey
…
```

```
…
/* ---------------------------------------------------------------------
 * Update some Variables before displaying the Menu
 * ---------------------------------------------------------------------
 */
Updvar:
MDate=date()          /* assuming MDATE/MTIME are defined in the MENU */
MTime=time('L')
Return
```

Example: Before Display update some variables via a callback procedure, and check command line input via an enter-exit

```
rckey=FSSMENU('$DISPLAY','UPDVAR','CHECKKEY')
say 'End Key 'rckey
…
…
/* ---------------------------------------------------------------------
 * Update some Variables before displaying the Menu
 * ---------------------------------------------------------------------
 */
Updvar:
MDate=date()          /* assuming MDATE/MTIME are defined in the MENU */
MTime=time('L')
Return
/* ---------------------------------------------------------------------
 * Check user's Input in command Line
 *    Return code handling:
 *    0   input has been handled by exit, re-display Menu
 *    4   input has not been handled, continue with internal checks
 *    8   exit Menu immediately
 * ---------------------------------------------------------------------
 */
CheckKey:
Parse arg actionkey,usercommmand
If length(usercommand)>2 then do
   Say usercommand' is not an Option'
   Return 0  /* continue, command already checked */
End
Return 4     /* maybe an Option, continue to option check */
```

5.7 FMTMENU Fully Defined Menu Screens

Using FSSMENU, you can define the menu lines and generate the menu handling, but it must be incorporated in a normal REXX script containing the other parts of the screen definition and handling.

FMTMENU allows you the definition of a menu screen in one step, but there are additional screen definitions in the menu possible.

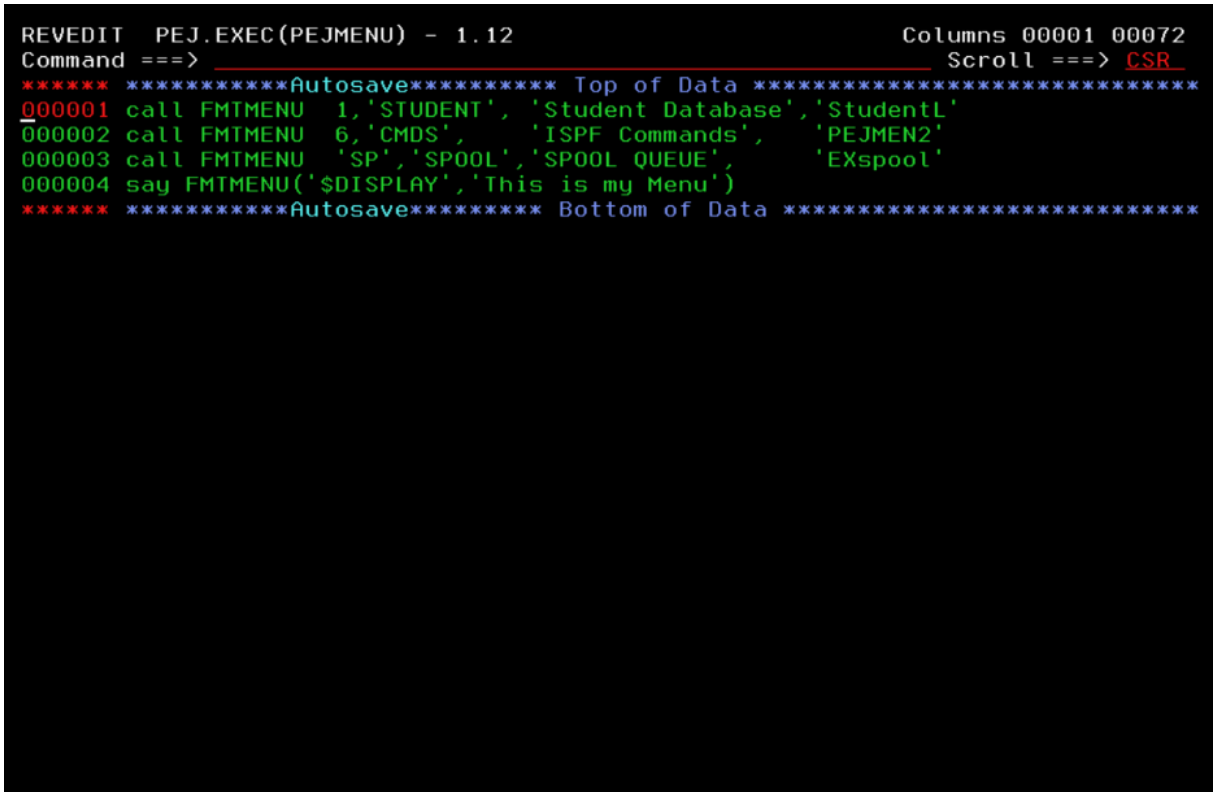
5.7.1 Definition of the Menu

```
CALL FMTMENU 'option','note','description','rexx-script'
```

option option code which leads to performing the associated action. The option can be a numeric or alphanumeric string.
note the short description of the action to perform
description long description of the action to perform
rexx-script REXX script which performs the action when the option is selected. Note the difference, to FSSMENU, here it must be a REXX script, but it may also contain calls to TSO, etc.

An FMTMENU always contains a title line (first row) an option line (second row) a message line (last row -1) and a footer line (last row).

5.7.2 Example Menu definition:

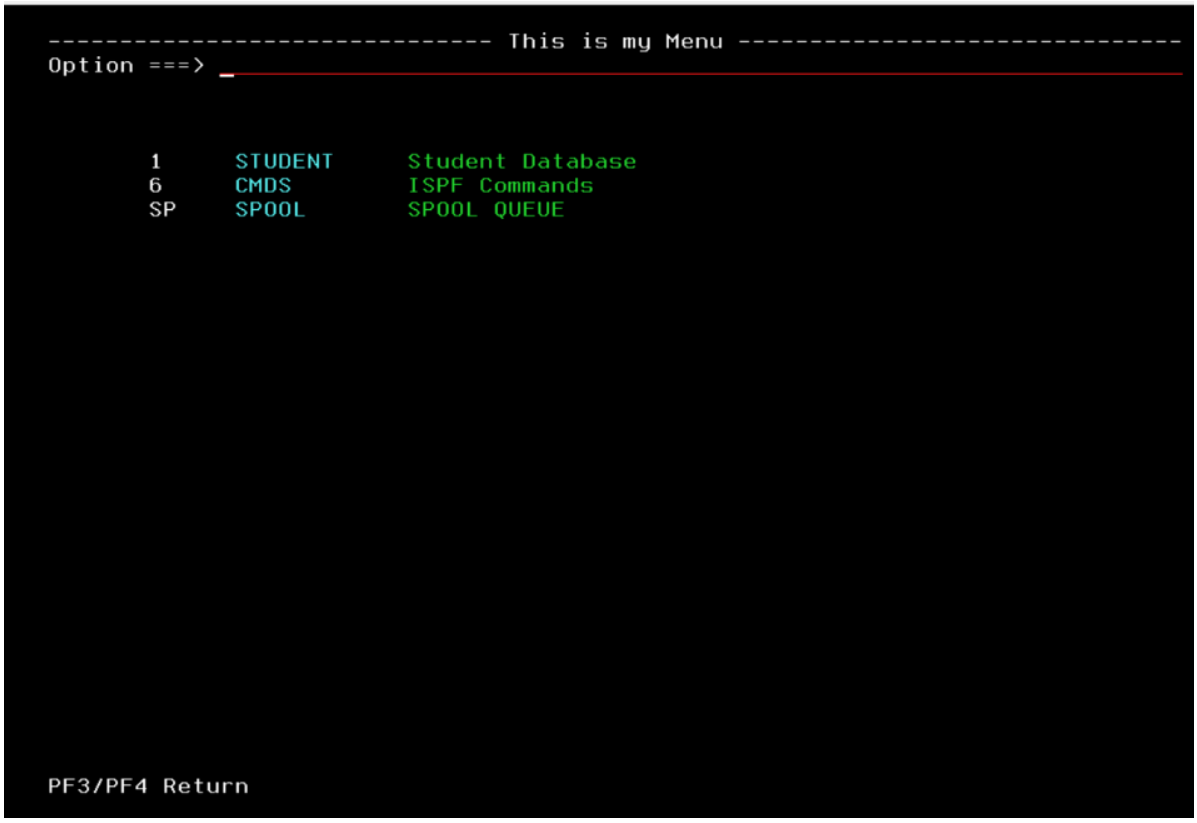


5.7.3 Displaying the FMTMENU Screen

To display the menu and handle the selected actions, FMTMENU must be called with the $DISPLAY parameter:

```
returnkey=FMTMENU('$DISPLAY','menu-title'>)
```

returnkey key which was pressed to end the dialogue handling, it is either PF03, PF04, PF15, or PF16
$DISPLAY Display the menu defined before
menu-title defining the menu title



5.8 Menu Tailoring

There are some settings, which allow you to tailor the menu layout. The usage of the stem _screen defines all settings **.xxx.** These settings are supported in FSSMENU as well as in FMTMENU.

_screen. MenuRow starting row of first Menu entry (default is 4)
_screen.MenuCol Column of Option parameter (default is 6)
_screen.Menucol2 Column of note parameter (default is _screen.MenuCol+3)
_screen.Menucol3 Column of note parameter (default is _screen.MenuCol+14)
Note for FSSMENU: there are separate parameters **startrow** and **startcol** in the menu definition:

```
CALL FSSMENU 'option','note','description','action',[startRow],[startCol]
```

If they are defined, they take precedence over the screen.MenuRow and screen.MenuCol definition.

_screen.MenuFooter defines the contents of a footer line (placed on the last row)

Setting just for FSSMENU (in FMTMENU they are managed automatically)
_screen.MenuOption 1 adds an Option line, else it must be defined manually
_screen. MenuMessage 1 adds a message line (last row-1)
_screen. Menutitle' 1 adds a title line

5.9 Formatted List Output

The usage of SAY statements displays the standard output of a REXX script. The disadvantage you can not scroll in it. Alternatively, you can write it in a sequential file and view it after the script has ended.

By using the FMTLIST command and passing a result buffer in a stem variable, you can browse in the output while your REXX script is still running.

Example REXX reads entire RXDATE Member and displays it:

```
/* REXX */
ADDRESS TSO
 "ALLOC FILE(INDD) DSN('BREXX.RXLIB(RXDATE)')"
 "EXECIO * DISKR INDD (STEM Buffer."
 "FREE FILE(INDD)"
call fmtlist
return
```

```
CMD ==> _____ ROWS 00001/00191 COL 001 B01
***** ********************************* Top of Data **********************************
00001 /* ------------------------------------------------------------------------
00002  *  RXDATE Transforms Dates in various types
00003  *  ............................. Created by PeterJ on 21. November 2018
00004  *  RXDATE(<output-format>,<date>,<input-format>)
00005  *  date is formatted as defined in input-format
00006  *    it defaults to today's date
00007  *  Input Format represents the input date format
00008  *    it defaults to 'EUROPEAN'
00009  *     Base       is days since 01.01.0001
00010  *     JDN        is days since 24. November 4714 BC
00011  *     Julian     is yyyyddd    e.g. 2018257
00012  *     European   is dd/mm/yyyy e.g. 11/11/2018
00013  *     German     is dd.mm.yyyy e.g. 20.09.2018
00014  *     USA        is mm/dd/yyyy e.g. 12.31.2018
00015  *     STANDARD   is yyyymmdd   e.g. 20181219
00016  *     ORDERED    is yyyy/mm/dd e.g. 2018/12/19
00017  *  Output Format represents the output date format
00018  *    it defaults to 'EUROPEAN'
00019  *     Base       is days since 01.01.0001
00020  *     JDN        is days since 24. November 4714 BC
00021  *     Julian     is yyyyddd    e.g. 2018257
00022  *     Days       is ddd days in this year e.g. 257
00023  *     Weekday    is weekday of day e.g. Monday
00024  *     Century    is dddd days in this century
00025  *     European   is dd/mm/yyyy e.g. 11/11/2018
00026  *     German     is dd.mm.yyyy e.g. 20.09.2018
00027  *     USA        is mm/dd/yyyy e.g. 12.31.2018
00028  *     SHEurope   is dd/mm/yy   e.g. 11/11/18
00029  *     SHGerman   is dd.mm.yy   e.g. 20.09.18
00030  *     SHUSA      is mm/dd/yy   e.g. 12.31.18
```

Figure 1 Created list buffer

Using the PF7 and PF8 you scroll upward and forward, PF10 and PF11 scroll left and right.
M in the CMD line and PF7 moves buffer to the top, M and PF8 to the bottom.
A number and PF7 or PF8 moves the buffer the specified lines up or down.

5.9.1 FMTLIST Prerequisites

FMTLIST always displays the content of the stem variable BUFFER. The buffer must have the general structure:

BUFFER.0 contains the number of entries in BUFFER
BUFFER.1 contains the first line
BUFFER.2 second line

…

BUFFER.n last line

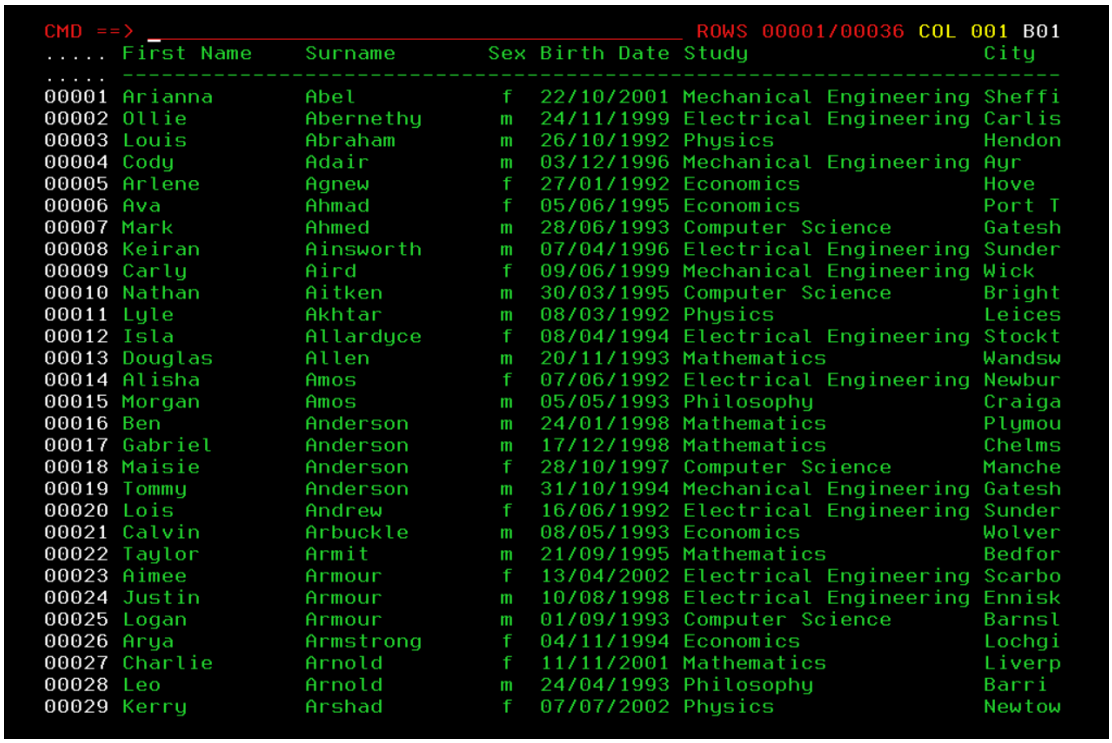
Alternatively, you can also display a String Array. Then you need to specify, in BUFFER.0:

`BUFFER.0="SARRAY "array-number`

5.9.2 FMTLIST calling Syntax

FMTLIST [length-line-area],[line-area-character],[header-1],[header-2],[applicationID]

length-line-area	length of displayed line-area, default is 5
line-area-character	character which should be displayed in the line area, default is none, then the line area contains the line number
header-1	this is an optional header line which is shown as first-line the displayed buffer
header-2	optional second header, only if header-1 is also defined
applicationID	If you specify an application ID, the FMTLIST screen supports line commands. The Line commands must be defined and coded in the calling REXX script as a callback label: applicationID_linecommand .



```
CMD ==> _                                          ROWS 00001/00036 COL 001 B01
..... First Name    Surname       Sex Birth Date Study                 City
..... ------------------------------------------------------------------------
00001 Arianna       Abel           f  22/10/2001 Mechanical Engineering Sheffi
00002 Ollie         Abernethy      m  24/11/1999 Electrical Engineering Carlis
00003 Louis         Abraham        m  26/10/1992 Physics                Hendon
00004 Cody          Adair          m  03/12/1996 Mechanical Engineering Ayr
00005 Arlene        Agnew          f  27/01/1992 Economics              Hove
00006 Ava           Ahmad          f  05/06/1995 Economics              Port T
00007 Mark          Ahmed          m  28/06/1993 Computer Science       Gatesh
00008 Keiran        Ainsworth      m  07/04/1996 Electrical Engineering Sunder
00009 Carly         Aird           f  09/06/1999 Mechanical Engineering Wick
00010 Nathan        Aitken         m  30/03/1995 Computer Science       Bright
00011 Lyle          Akhtar         m  08/03/1992 Physics                Leices
00012 Isla          Allardyce      f  08/04/1994 Electrical Engineering Stockt
00013 Douglas       Allen          m  20/11/1993 Mathematics            Wandsw
00014 Alisha        Amos           f  07/06/1992 Electrical Engineering Newbur
00015 Morgan        Amos           m  05/05/1993 Philosophy             Craiga
00016 Ben           Anderson       m  24/01/1998 Mathematics            Plymou
00017 Gabriel       Anderson       m  17/12/1998 Mathematics            Chelms
00018 Maisie        Anderson       f  28/10/1997 Computer Science       Manche
00019 Tommy         Anderson       m  31/10/1994 Mechanical Engineering Gatesh
00020 Lois          Andrew         f  16/06/1992 Electrical Engineering Sunder
00021 Calvin        Arbuckle       m  08/05/1993 Economics              Wolver
00022 Taylor        Armit          m  21/09/1995 Mathematics            Bedfor
00023 Aimee         Armour         f  13/04/2002 Electrical Engineering Scarbo
00024 Justin        Armour         m  10/08/1998 Electrical Engineering Ennisk
00025 Logan         Armour         m  01/09/1993 Computer Science       Barnsl
00026 Arya          Armstrong      f  04/11/1994 Economics              Lochgi
00027 Charlie       Arnold         f  11/11/2001 Mathematics            Liverp
00028 Leo           Arnold         m  24/04/1993 Philosophy             Barri
00029 Kerry         Arshad         f  07/07/2002 Physics                Newtow
```

Figure 2 Example of FMTLIST with 2 header lines:

If you use PF7/PF8 to scroll up and down, the two header lines are always displayed as the buffer top lines.

FMTLIST supported PF Keys and Scrolling commands

PF3/PF4 exit FMTLIST screen
PF7 scroll one page up
PF8 scroll one page down
PF10 shift buffer 50 columns left
PF11 shift buffer 50 columns right

PF12 Display last command

If you use a combination of a number in the command line and PF7 or PF8, the buffer scrolls the number of lines up or down.
Command-line functions

TOP	displays the first line of the buffer
M and PF7	displays the first line of the buffer
BOTTOM	displays the last line of the buffer
BOT	displays the last line of the buffer
M and PF8	displays the last line of the buffer

5.9.3 FMTLIST Customising Options

By setting _SCREEN.xxxx, you can manipulate the appearance of FMTLIST in various ways:

Variable Name	Default	Allowed Values	Note
_screen.cmdchar	blank		Command Line character building the command line. The default is blank and creates an empty command line which is displayed with the 3270 attribute #USCORE If you set it as BLANK (keyword) then the command line is empty and #USCORE is not used.
_screen.color.Cmd	#red	3.4.3 Attribute Definitions	Colour of Command Line
_screen.color.header1	#blue	3.4.3 Attribute Definitions	Colour of the first header line (if defined)
_screen.color.header2	#blue	3.4.3 Attribute Definitions	Colour of the second header line (if defined)
_screen.color.Stats	#white	3.4.3 Attribute Definitions	Colour of Statistics (line and buffer numbering)
_screen.color.Top1	#red	3.4.3 Attribute Definitions	Colour of line area first line
_screen.color.Top2	#blue	3.4.3 Attribute Definitions	Colour of line conten first line (Top of Data)
_screen.color.Bot1	#red	3.4.3 Attribute Definitions	Colour of line area last line
_screen.color.Bot2	#blue	3.4.3 Attribute Definitions	Colour of line content last line (End of Data)
_screen.color.List1	#white	3.4.3 Attribute Definitions	Colour of line area (content part)
_screen.color.List2	#green	3.4.3 Attribute Definitions	Colour of line content part
_screen.footer	undefined	Content of footer (PF1 …)	Fixed Footer Line (at screen height)
_screen.color.footer	#white	3.4.3 Attribute Definitions	Colour of line content part

_screen.Primary	1	0 / 1	0 disabling user primary commands 1 any primary command is allowed
_screen.Message	undefined	1 for defining message	Fixed Message Line (screen height-1)
_screen.TopRow	1	1 up to Screen height-3	Begin row of fmtlist, if it is 2 or more there are empty lines above FMTLIST
_screen.TopRow.proc	Undefined		Is a call-back proc name in the REXX calling FMTLIST. There you can define the line above the FMTLIST screen. They can be set with FSSText commands. The number of added rows must not exceed _screen.TopRow-1
_screen.BotLines	Lines reserved at bottom of FMTLIST	1 up to Screen height-3	As screen height is dynamic depending on the 3270 definitions.
_screen.BotLines.proc	Undefined		Is a call-back proc name in the REXX calling FMTLIST. There you can define the lines at the end of the FMTLIST screen. They can be set with r FSSText commands. The first line number which can be set is passed as arg(1) parameter. For consistency reasons of call back parameters, it is enclosed in quotes. This means you must strip them off: `first=strip(translate(arg(1),'',"'"))`

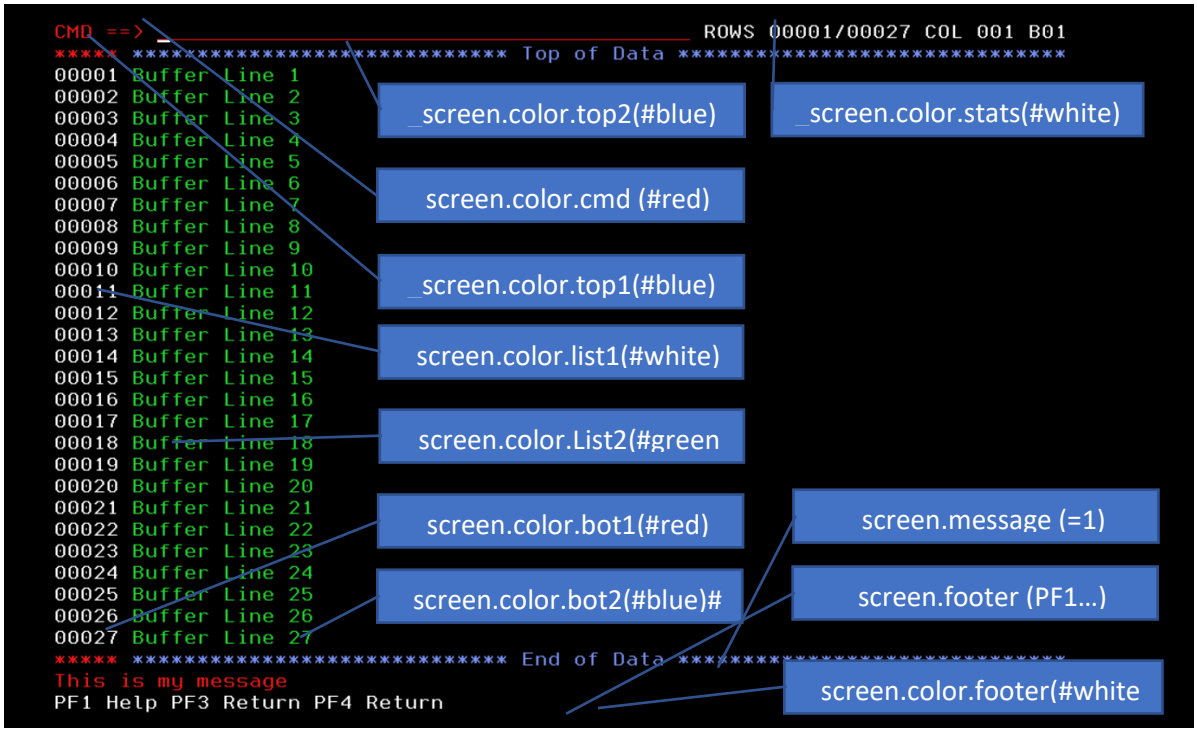


Figure 3 Settings related _screen.xxx Variables

5.9.4 FMTLIST calling other REXX scripts from the command line

If you want to play another REXX script from within the FMTLIST buffer you can do so, by entering:

rexx-script-name in the command line

Simple REXX scripts

A simple Rexx script does not contain any call to an FSS Screen. A sequence of say statements may provide the result, or you can place it in a buffer.x stem. If you do so, the result displayed in the current FMTLIST buffer. Which means the existing content is overwritten.

```
Buffer.1='first line'
Buffer.2='second line'
Buffer.0=2
```

If you want to keep the contents of the current buffer, use the prefix command **LOOKASIDE rexx-script-name**, and a new stacked buffer is created residing on top of the previous buffer.
The previous buffer can be re-activated by pressing the PF3 key; it destroys the current buffer and returns to the last buffer.

If the called rexx-script contains an FMTLIST, FSSMENU, or FMTMENU itself a new buffer is created automatically.

5.9.5 Formatted List Line and Primary Commands

The FMTLIST Buffer supports Line Commands if it is called with an applicationID. The line command is coded within the calling procedure (performing the FMTLIST) as a callback label, to keep the scope of the variables there must not be a PROCEDURE statement used. The callback label must be coded as:

applicationID_linecommand. In the following example there is a line command **S**, **U**, and **D** defined :

```
/* REXX */
ADDRESS TSO
 "ALLOC FILE(INDD) DSN('BREXX.RXLIB(RXDATE)')"
 "EXECIO * DISKR INDD (STEM Buffer."
 "FREE FILE(INDD)"
call fmtlist ,,,,MYLIST /* MYLIST is application ID */
return
/* ---------------------------------------------------------------------
 * Line commands are organised as "call-back' labels to the calling REXX
 *   Format is REXX name_linecmd
 * ---------------------------------------------------------------------
 */
mylist_s:     /* line command S, just output selected line  */
  say Arg(1)
return 0      /* tell FMTLIST to proceed normally     */
mylist_u:     /* line command U, allow editing line   */
  newLine=lineedit(,arg(1))
return 4      /* tell FMTLIST, you changed line       */
mylist_e:     /* line command E, automatically change line
*/
  newLine='new Line set'
  zerrsm='update'
  zerrlm='Line has been updated'
```

```
return 4      /* tell FMTLIST, line is changed line     */
mylist_d:     /* Delete Line */
  return 5    /* tell FMTLIST to delete selected line
*/
```

RC Code actions

RC=0 means the line command was processed

RC=4 means the line command was processed; if the REXX variable **NEWLINE** contains a value, the selected line will be overwritten by this value.

RC=5 delete this line

RC=6 a completely **new buffer.n** stem has been provided and should be displayed immediately. The old buffer content will be removed. If you set a ZERRSM or ZERRLM message the message will be kept and displayed.

RC=7 a **new buffer.n** stem has been provided and should be displayed in a new FMTLIST buffer, which is stacked on top of the previous one. Once you return with PF3 you will see the old buffer content. If you set a ZERRSM or ZERRLM message the message will be kept and displayed.

RC=8 invalid line command

Additionally, you can change the colour of the line in the buffer; you have to set:

SETCOLOR1 sets the colour of the selected line of the line area, e.g. setcolor1=#green

SETCOLOR2 sets the colour of the selected buffer content line, e.g. setcolor2=#red

If none or just one of the colours have been set, the other field colour remains unchanged

5.9.6 Formatted List Special Call-Back labels

FMTLIST supports certain call-back labels (defined in the calling REXX) if FMTLIST is called with an applicationID.

HELP

This example shows the definition of a help system for the volume list REXX. The applicationID is **VOLUMES**, therefore the call-back label is **volumes_help**:

```
000013 call fmtlist ,,copies(' ',20)'Volumes of your MVS3.8','Volume  Unit Device','VOLUMES'
000014 return 0
000015 /* ------------------------------------------------------------
000016  * VOLUMES Help
000017  * ------------------------------------------------------------
000018  */
000019 volumes_help:
000020   buffer.0=2
000021   buffer.1='This is my help for X34'
000022   buffer.2='oops, it is not yet defined'
000023   call fmtlist
000024 return 0
```

5.9.7 Formatted List Special labels

FMTLIST also supports calling generic procedures. They must be explicitly activated to be called. The location is of your choice, they can be defined in the calling REXX or as independent REXX.

TOPROW Procedure

Allows you to embed an FMTLIST screen into a frame of your own. It must be activated by defining the beginning position of the FMTLIST screen, and the label which creates the top-line content. The Header must be provided with FSS Text definitions. It is not (yet) intended to allow input fields.

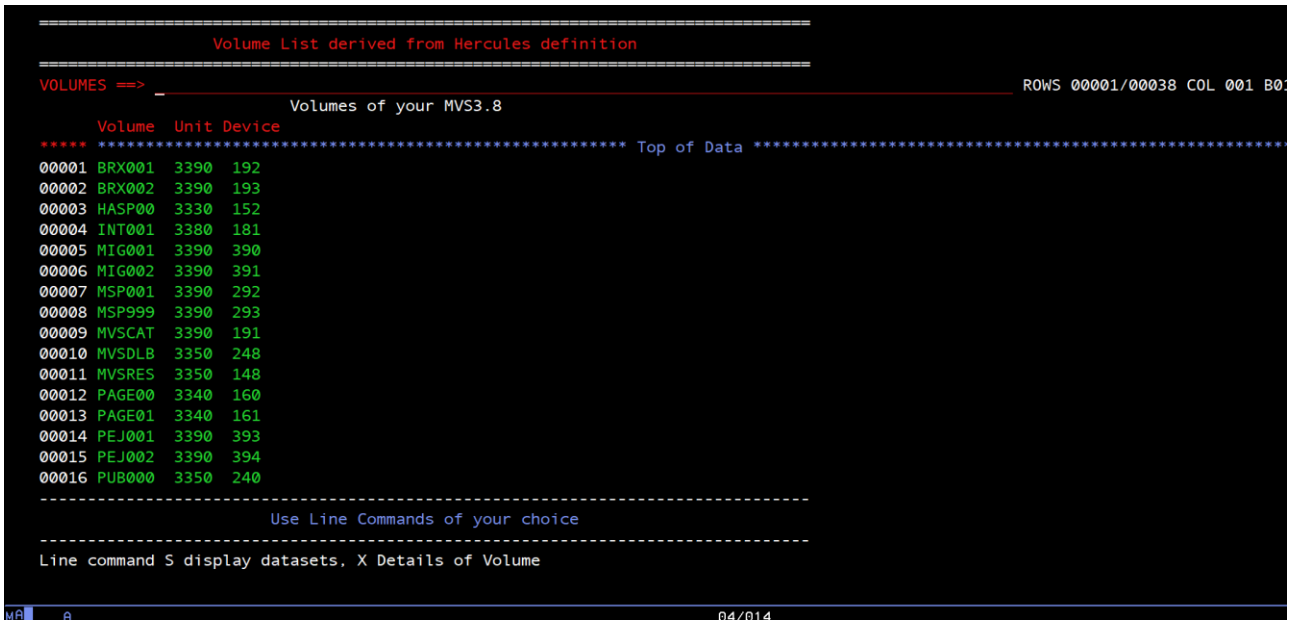
BOTLINES Procedure

Allows you to embed an FMTLIST screen into a frame of your own. It must be activated by defining the bottom lines of the FMTLIST screen, and the label which creates the bottom lines content. It is not (yet) intended to allow input fields.

The following example shows the definition of a frame consisting of 3 header and footer lines:

```
_screen.TopRow=4
_screen.TopRow.Proc="x34Header"
_screen.BotLines=3
_screen.BotLines.proc="X34Footer"
call fmtlist ,,copies(' ',20)'Volumes of your MVS3.8','Volume  Unit
Device','VOLUMES'
return 0
/* ------------------------------------------------------------
 * VOLUMES Frame Header
 * ------------------------------------------------------------
 */
x34Header:
  delim=copies("=",80)
  hdr  =Center("Volume List derived from Hercules definitions",80)
  Address FSS
  'TEXT 1 2 #PROT+#HI+#White delim'
  'TEXT 2 2 #PROT+#HI+#RED hdr'
  'TEXT 3 2 #PROT+#HI+#White delim'
return 0
/* ------------------------------------------------------------
 * VOLUMES Frame Footer
 * ------------------------------------------------------------
 */
x34Footer:
  delim=copies("-",80)
  cmt  =Center("Use Line Commands of your choice",80)
  Address FSS
  'TEXT 24 2 #PROT+#HI+#White delim'
  'TEXT 25 2 #PROT+#HI+#BLUE cmt'
  'TEXT 26 2 #PROT+#HI+#White delim'
return 0
```

Result:



5.9.8 Formatted List Samples

There are several scripts in BREXX.V2R5M2.SAMPLES illustrating the usage of FMTLIST.

FMTOPBOT	has an embedded FMTLIST with a user-defined header and footer lines.
@STUDENTL	the front end of the VSAM student database example
#BROWSE	Displays the LISTALC command

5.10 Debugging Simple Screen Applications

If you need to debug the behaviour of simple screen applications, you can switch on a trace feature in the calling REXX script:

_screen.FTRACE=1

You get a trace of the performed step within the screen application .

```
/* REXX */
do i=1 to 35
   buffer.i='Buffer Line 'i
end
buffer.0=i-1
/*
_screen.color.top2=#yellow
_screen.color.mylist=#red
_screen.color.cmd  =#blue
_screen.color.stats=#white
*/
_screen.footer='PF1 Help PF3 Return PF4 Return'
```

```
_screen.Message=1
CALL FMTLIST  ,,'','','TEST'
```

Displaying Trace in TSO

```
09:45:27.09 Entering FMTLIST
09:45:27.18 Display Screen
 ***
The screen is displayed, waiting for the next user action

09:45:56.65 User Action PF08
09:45:56.69 Command Line ''
09:45:56.71 Display Screen
 ***
The screen is displayed, waiting for the next user action

09:46:42.13 User Action PF07
09:46:42.17 Command Line '10'
09:46:42.20 Display Screen
 ***
The screen is displayed, waiting for the next user action

09:47:10.09 User Action PF03
09:47:10.09 Command Line ''
09:47:10.09 Leaving FMTLIST
 ***
```

5.11 Formatted List Monitor FMTMON

By setting up a formatted list monitor you can monitor certain events on a timely basis. You can for example continuously view updated entries of the Master Trace Table

Example in **BREXX. V2R5M2.SAMPLE :**

```
CALL IMPORT FSSAPI
/* ---------------------------------------------------------------------------
 * FMTMON is an FSS application that refreshes itself every xxx milliseconds
 *    the refresh takes place in the call-back procedure MonTimeOut it must
 *    provide a new buffer or just return
 *    There is also an enter-key call-back procedure MonEnter where you can
 *    execute commands, e.g. CONSOLE and modify the buffer if wanted
 * ---------------------------------------------------------------------------
 */
 call fmtmon "MVS Trace Table",1000
return 0
```

5.11.1 FMTMON calling Syntax

FMTMON header,[refresh-frequency]

header is displayed as title in the FMTMON screen

refresh-frequency refresh timer in milliseconds

5.11.2 FMTMON Call-Back Procedures

FMTMON requires two call-back procedures, which must be implemented in the calling REXX procedure.

1. **MONENTER**: is called when has entered input and presses the enter-key

```
/* ---------------------------------------------------------------
 * MONENTER Call Back PROC of FMTMON  Enter key pressed, do something
 *   return 0  continue normally
 *          4  continue normally, buffer is not touched
 *          8  end monitor (as PF3)
 *         12  end monitor (as PF4)
 * ---------------------------------------------------------------
 */
MonEnter:
  call CONSOLE arg(1)   /* action requested console command */
return 0
```

2. **MONTIMEOUT**: is called when the frequence-time-out has been reached

```
/* ---------------------------------------------------------------
 * MONTIMEOUT  Call Back PROC of FMTMON  Enter key pressed, do something
 *          Timeout in FSS, you can provide new content in
 *          BUFFER.i i=1 to number of lines
 *          BUFFER.0 must contain number of lines
 *   return 0  continue buffer is unchanged
 *          1  continue new buffer provided
 * ---------------------------------------------------------------
 */
MonTimeout:     /* arg(1) entry count */
/* create new contents of FMTMON Buffer.
return
```

5.11.3 FMTMON provide data to display

FMTMON displays the content of the stem variable BUFFER, typically it is updated in the MONTimeout call-back procedure.

The buffer must have the general structure:

BUFFER.0 contains the number of entries in BUFFER
BUFFER.1 contains the first line
BUFFER.2 second line
…
BUFFER.n last line

As the name is fixed, it does not need to be passed to FMTMON.

5.11.4 FMTMON predefined Action Keys

Help key: PF1

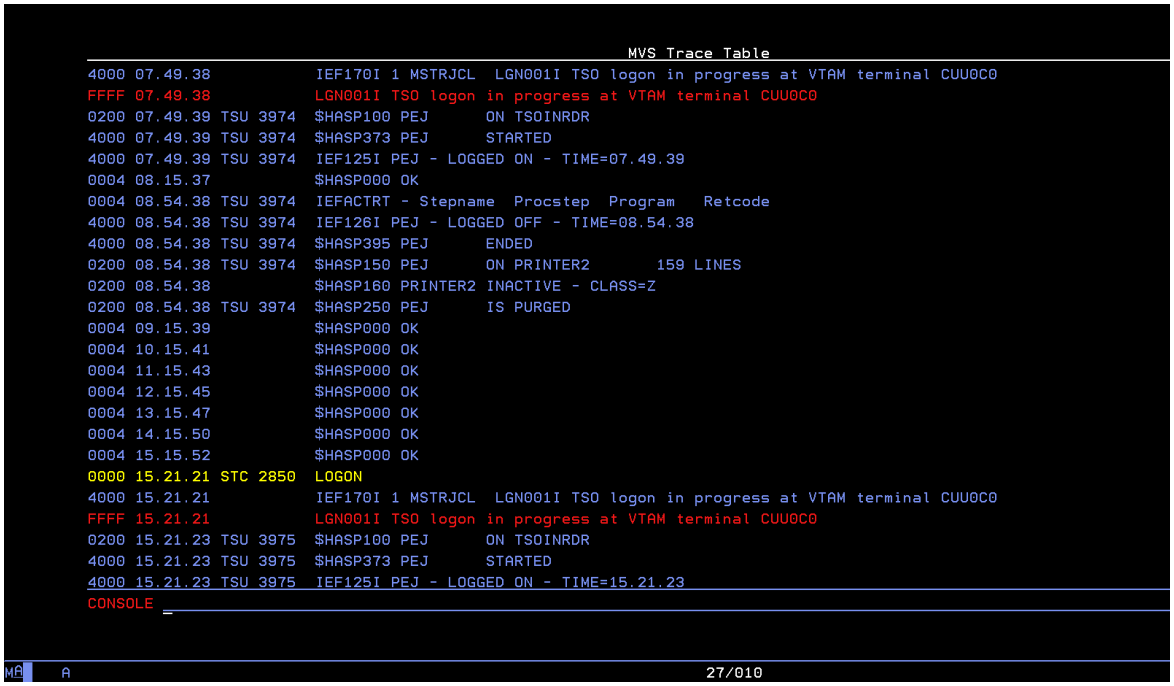
Scrolling keys: PF7/PF8

Commands: TOP/BOT/UP n(-lines)/DOWN n(-lines)

5.11.5 FMTMON Application display Master Trace Table

This example is stored in:

BREXX.V2R5M2.SAMPLES(MTT)



6 FSS Functions as Host Commands

Alternatively to the FSS functions described in ”FSS Function Overview” you can use the FSS Host command API directly. In this case, all definitions, calculations, validations, etc. must be handled by your REXX script directly.

6.1 INIT FSS Environment

Initialise the FSS environment; this must be performed before any other FSS call.

```
ADDRESS FSS
'INIT'
```

6.2 Defining a Text Entry

```
ADDRESS FSS
'TEXT 'row column attributes text'
```

text: text to be displayed on the screen
row: row where text should be placed
column: column where text should be placed.

attributes: screen attributes, like colours, protected, high-lighted etc. For details refer to the attributes section

6.3 Defining a Field Entry

```
ADDRESS FSS
'FIELD  'row column attributes field flen [preset]'
```

text: text to be displayed on the screen
row: row where text should be placed
column: column where text should be placed.
attributes: screen attributes, like colours, protected, high-lighted etc. For details refer to the attributes section
field: Screen field name
flen: length of input area representing field name
preset: content initially displayed (optional), defaults to blank

6.4 Getting Field Content

```
ADDRESS FSS
'GET FIELD field rexx-variable'
```

field: Screen field name
rexx-variable: variable receiving the field content

6.5 Setting Field Content

```
ADDRESS FSS
'SET FIELD field value'
     or
'SET FIELD field 'rexx-variable
```

field: Screen field name
value new field content
rexx-variable: variable containing the field content

6.6 Setting Cursor to a field

Sets the cursor to the beginning of the Screen Field

```
ADDRESS FSS
'SET CURSOR field'
```

field: Screen field name

6.7 Setting Colour

Sets the Colour of a Screen Field

```
ADDRESS FSS
  'SET COLOR field/text colour'
```

field: Screen field name
colour: Color definition,for details refer to the attributes section

6.8 Getting action Key

When the user presses an action-key on a screen, the key value can be fetched in a rexx-variable

```
ADDRESS FSS
'GET AID rexx-variable'
```

rexx-variable: variable receiving the action key

6.9 Display or Refresh Formatted Screen

Used to display the Formatted Screen the first time, or to refresh an active screen

```
ADDRESS FSS
'REFRESH'
```

6.10 End or Terminates FSS Environment

Ends the Formatted Screen environment and releases all used main storage.

```
ADDRESS FSS
```

```
'TERM'
```

6.11 Get Terminal Width

```
ADDRESS FSS
'GET WIDTH rexx-variable'
```

rexx-variable: variable receiving the action key

6.12 Get Terminal Height

```
ADDRESS FSS
'GET HEIGHT rexx-variable'
```

rexx-variable: variable receiving the action key

BREXX/370 V2R5M2 Formatted Screens

Table of Contents

BREXX/370 V2R5M2 Formatted Screens