BREXX/370 V2R2M0 Formatted Screens

Document Version 1.0
Authors: Peter Jacob (pej), Mike Großmann(mig)

The following document is a brief description of the new Formatted Screen (FSS) feature. It allows to set up 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 using SAYs
#FSS1COL	A pre-packed FSS application to generate input requests (in one columns)
#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 call-back to test user’s input

2 FSS Limitation

FSS supports screen buffers up to 4096 bytes. If your screen size exceeds it, FSS take advantage of the maximum of 4096 bytes and reduces the buffer size to it. The buffer size is calculated by the maximum number of columns multiplied by the maximum number of lines of your terminal emulation.

If you have an 80*24 screen size the buffer size is 1920 bytes. For 160*62, the size is 9920 bytes which exceed the maximum size of 4096. Which means the maximum columns and/or the maximum lines are reduced to display the defined FSS Screen. This reduction does not impact other applications which take advantage of the full size of your screen.

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 in 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 initialize :

```
/* 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 prior to any other FSS call.

CALL FSSINIT

3.2 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

3.3 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.

3.3.1 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 real text or field-definition start at column+1.

3.3.2 Important Notice on Screen Definitions

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

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

3.3.3 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 will be displayed high-lighted
#NON	text/field-input is invisible
#BLINK	text/field blinks
#REVERSE	back ground is set with defined colour text appears white
#USCORE	Underscore field

#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

3.4 FSSTITLE   Displays a centred Title in Screen line 1

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

3.5 FSSOPTION   Create OPTION Line

Creates an OPTION 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

3.6 FSSCOMMAND Create a Command Line

Creates a COMMAND, typically used to allow entering commands.

```
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

3.7 FSSMESSAGE Create a Message Line

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

```
CALL FSSMSG [row[,attribute]]
```

row	defaults to 2
attribute	attribute of message line, default is #PROT+#HI+#RED

3.8 FSSZERRSM Set Error/Warning/Info Short Message

The message will be set in Field ZERRMS. ZERRMS will be 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’
```

3.9 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!

3.10 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!

3.11 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(…)

3.12 FSSCURSOR Set Cursor to a Field

```
CALL FSSCURSOR ‘field’
```

3.13 FSSCOLOUR Change Colour of a Field

CALL FSSCOLOUR 'field',colour-attribute alternatively

CALL FSSCOLOR 'field' ,colour-attribute

3.14 FSSKEY Return Key entered

When the user presses an action-key on a screen the key value can be accessed by FSSKEY

key=FSSKEY()

By FSS supported keys:

REXX Variable	numeric Value
#ENTER	125
#PFK01	241
#PFK02	242
#PFK03	243
#PFK04	244
#PFK05	245
#PFK06	246
#PFK07	247
#PFK08	248
#PFK09	249
#PFK10	122
#PFK11	123
#PFK12	124
#PFK13	193
#PFK14	194
#PFK15	195
#PFK16	196
#PFK17	197
#PFK18	198
#PFK19	199
#PFK20	200
#PFK21	201
#PFK22	74
#PFK23	75
#PFK24	76
#CLEAR	109
#RESHOW	110

3.15 FSSDISPLAY Display/Refresh a generated Formatted Screen

Displays or Re-Displays the active screen

CALL FSSDISPLAY or

CALL FSSREFRESH

3.16 Get Screen Dimensions

`WIDTH=FSSWidth()`	returns number of available columns defined by Emulation
`HEIGHT=FSSHeight()`	returns number of available rows defined by Emulation

3.17 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’,’short-description’,’long-description’,’action’`

option	option code which leads to perform the associated action
short-description	short description of action to perform
long-description	long description of action to perform
action	action will be performed is associated option is seleted the action must be prefixed by TSO for a TSO function call or with CALL if a REXX procedure should be called.

The FSS menu definitions can be included within a normal FSS Screen definition to add additional fields or text parts to the formatted screen. These parts can be dynamically updated if you specify a call-back procedure in the FSSMENU Display call.

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'
…
```

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

`returnkey=FSSMENU('$DISPLAY'<,call-back-procedure)`

returnkey	key which was pressed to end the dialogue handling, 243 is PF3, 244 is PF4,195 is PF15, 196 is PF16
$DISPLAY	Display the menu defined before
Call-back-procedure	optional own callback procedure (internal or external) to update FSS variables or other variables

Example:

```
rckey=FSSMENU('$DISPLAY','UPDATE')
```

```
say 'End Key 'rckey
```

3.18 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 Definitions

There is a simple way to create formatted screens using preformatted rexx scripts. 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 will create and display 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 will get the attributes in two columns

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

1. First Name  ===> ___________________ 2. Family Name ===> ________________
3. UserId      ===> ___________________ 4. Department  ===> ________________

```

Entered input will be 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 column formatted 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 will 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 it is not changed the input fields will 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 will be 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 next 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 call-back function:

_screen.CallBack='internal-subprocedure'

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

```
_screen.CallBack='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 ist mandatory'
     call FSSzerrlm 'Please enter valid content in Field 2'
     return 1
  end
…
```

In case of 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 call-back sub-procedure signals with its return code how to proceed:

return 0 : everything ok, leave screen an pass control back to calling rexx

any other return code then 0:

return 1 : something is wrong, short and error message (if set) are displayed

5.6 Formatted List Output

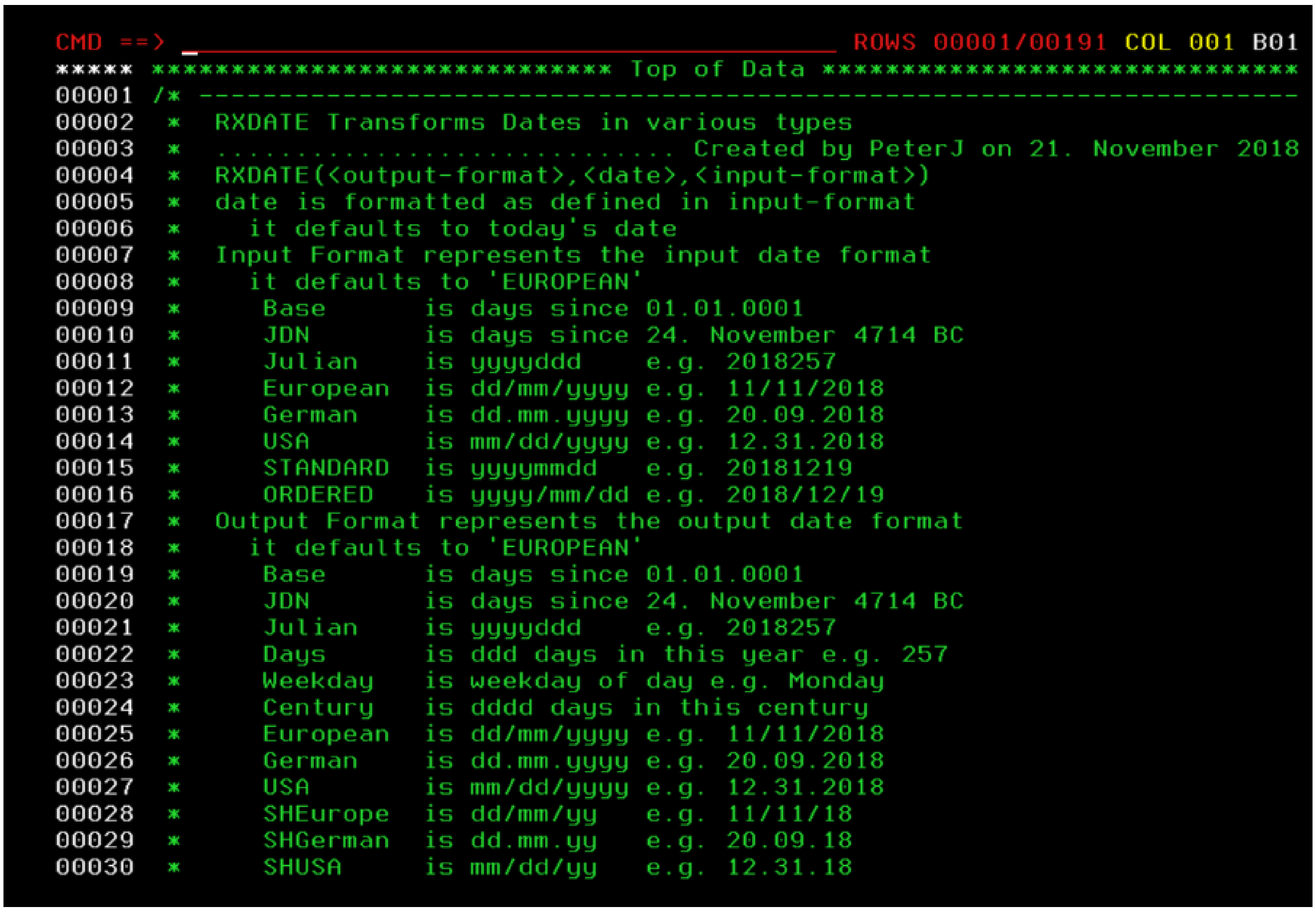
The normal output of a REXX script will be displayed by the usage of SAY statements. The disadvantage you can not scroll in it. Alternatively you can write it in a sequential file and view it after the script has ended.

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
```

Creates the following list buffer:



```
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
```

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.

5.6.1 FMTLIST Prerequisites

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

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

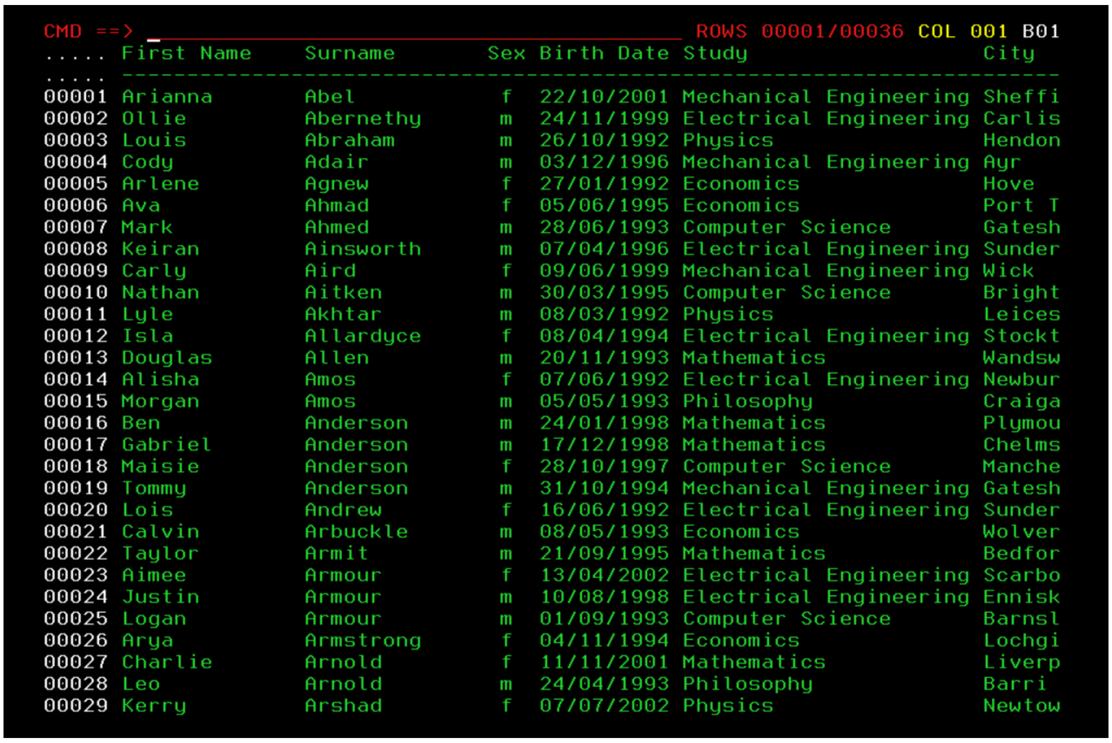
As the name is fixed, it does not need to be passed to FMTLIST.

5.6.2 FMTLIST calling Syntax

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

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

Example of FMTLIST with 2 header lines:



```
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
```

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

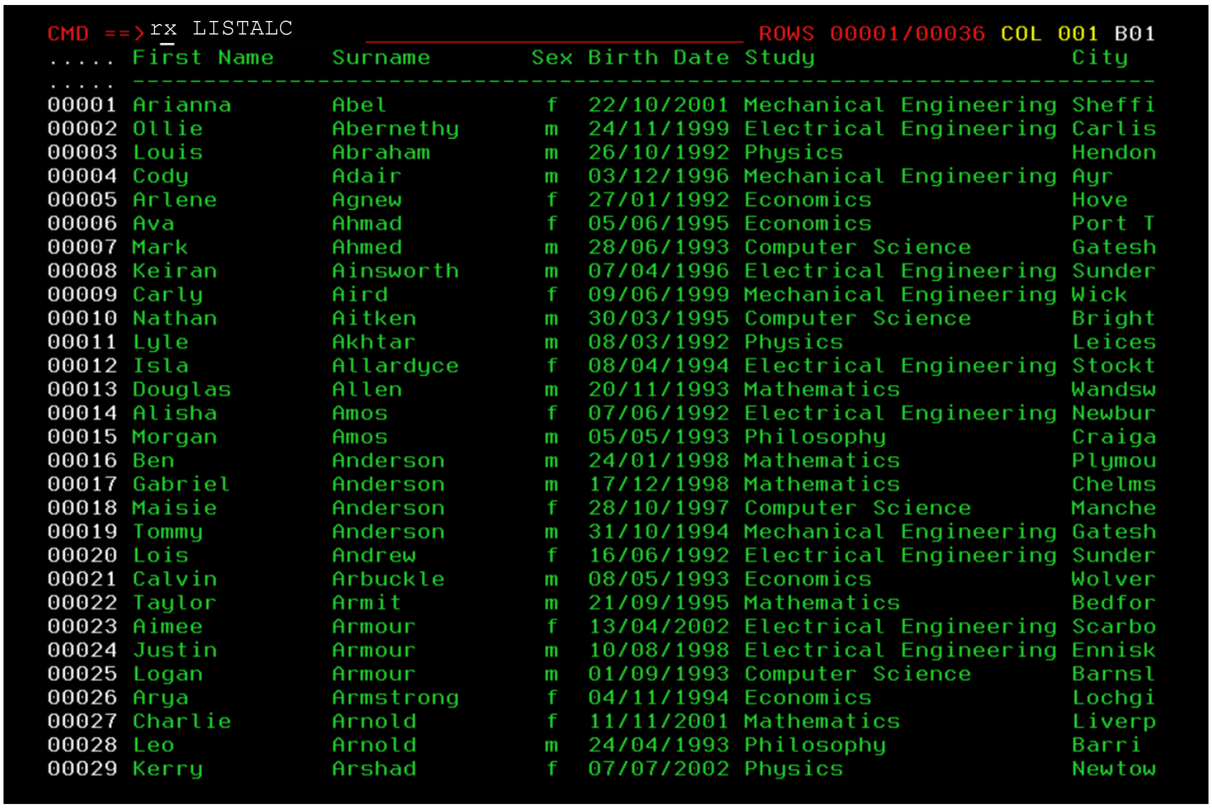
5.6.3 FMTLIST calling other REXX scripts from the command line

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

RX or REXX rexx-script-name in the command command line

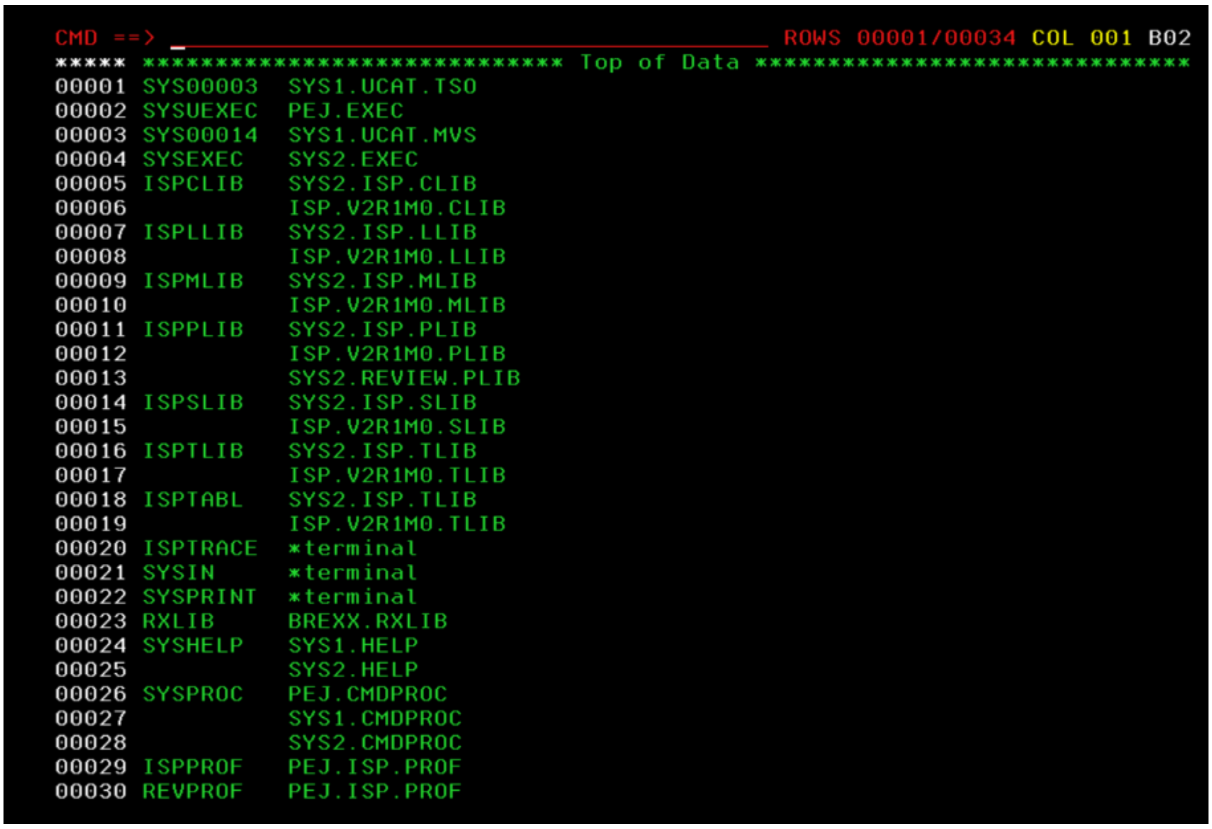
The buffer is saved internally and the specified REXX is called. The called REXX may itself use the FMTLIST method to output the result, or a simple sequence of say statements.
Once you leave the newly displayed buffer by PF3 you return to the FMTLIST buffer which was active.

Example:



```
CMD ==>rx LISTALC                                   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
```

The created FMTLIST buffer, in this case the output of the LISTALC command is created and displayed



```
CMD ==> _                                            ROWS 00001/00034 COL 001 B02
***** ***************************** Top of Data ******************************
00001 SYS00003  SYS1.UCAT.TSO
00002 SYSUEXEC  PEJ.EXEC
00003 SYS00014  SYS1.UCAT.MVS
00004 SYSEXEC   SYS2.EXEC
00005 ISPCLIB   SYS2.ISP.CLIB
00006           ISP.V2R1M0.CLIB
00007 ISPLLIB   SYS2.ISP.LLIB
00008           ISP.V2R1M0.LLIB
00009 ISPMLIB   SYS2.ISP.MLIB
00010           ISP.V2R1M0.MLIB
00011 ISPPLIB   SYS2.ISP.PLIB
00012           ISP.V2R1M0.PLIB
00013           SYS2.REVIEW.PLIB
00014 ISPSLIB   SYS2.ISP.SLIB
00015           ISP.V2R1M0.SLIB
00016 ISPTLIB   SYS2.ISP.TLIB
00017           ISP.V2R1M0.TLIB
00018 ISPTABL   SYS2.ISP.TLIB
00019           ISP.V2R1M0.TLIB
00020 ISPTRACE  *terminal
00021 SYSIN     *terminal
00022 SYSPRINT  *terminal
00023 RXLIB     BREXX.RXLIB
00024 SYSHELP   SYS1.HELP
00025           SYS2.HELP
00026 SYSPROC   PEJ.CMDPROC
00027           SYS1.CMDPROC
00028           SYS2.CMDPROC
00029 ISPPROF   PEJ.ISP.PROF
00030 REVPROF   PEJ.ISP.PROF
```

Pressing the PF3 key returns to the previous displayed buffer.

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 prior to 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.

```
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 V2R2M0 Formatted Screens

Table of Contents