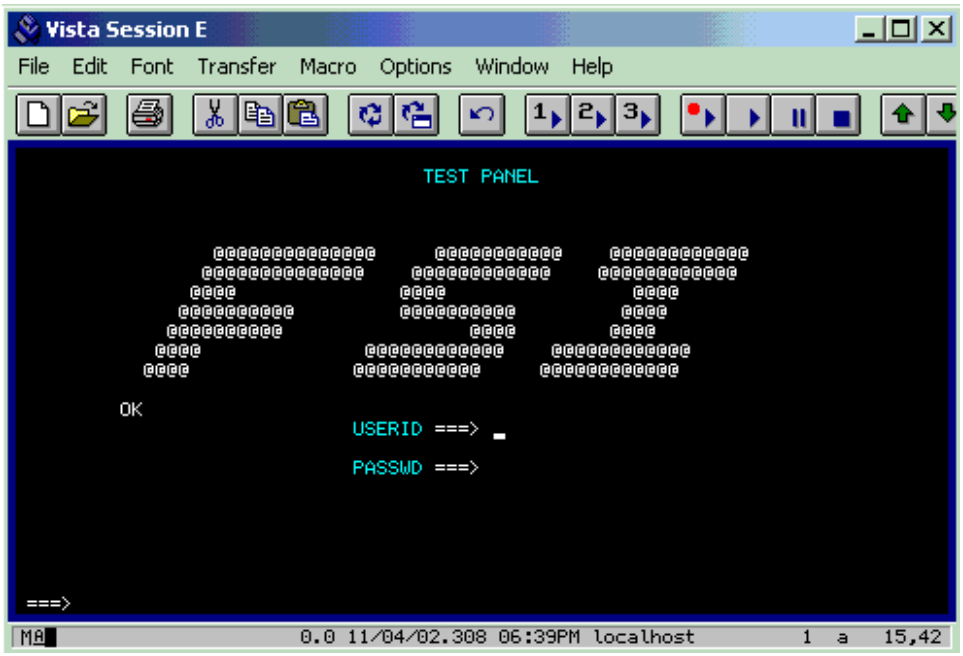
Full Screen Interface

Version 1 - Release 2



Full Screen Interface (FSI) is a programming library modeled after SPF that supports full screen 3270 programming. To use FSI, one more more application panels are coded and stored in the Panel Library or PLIB data set. The application program uses a Display function call to display the panel. Input / output variables are used to allow the application to display dynamic content to the screen and allow the application program to retrieve input data from the terminal.

FSI uses an execution environment that the application program must run underneath. To run a FSI application, the FSI environment must first be created using the FSISTART command specifying the application name as a parameter. The FSISTART command will load the FSI environment and then attach the application program.

Standard Disclaimer

FSI is a part-time fun project that I started working on while traveling. I found myself killing a lot of time in airports and fired up MVS 3.8 under Hercules on my laptop. The code "mostly works" at this point. There is a lot of need for improvement. Error handling is almost non-existent. I diagnose errors using the TSO TEST command since the code produces very few error messages.

There is a lot of room for enhancement and improvement if you would like to help take this project forward. When I get more spare time I may try to improve upon what I have here.

This is by far not the best code I have ever written in my life but it just evolved a few lines at a time. Since I was traveling, I did not have access to any of my reference materials. A lot of the original code was written to try something out and see what happened. When I get the right results, I added more code and attempted something else. If you look at the code it will pretty much look as though is was written that way. There was no overall plan or design I was working from.

Using FSI

Using FSI

It is assumed you understand JCL and the basics of Assembler programming in order to use the FSI API.

Getting FSI

FSI is available from my WEB Site as a stacked tape file created by IEBCOPY. You can get the tape image here: **fsi120.het**
or
fsi120.aws.zip

Installing FSI

You will need to use IEBCOPY to load the FSI files. Here is some sample JCL to load the FSI tape:

```
//FSILOAD JOB (5222),SPRINKLE,CLASS=A,MSGCLASS=A
//FSILOAD PROC PFX=FSI,VRM=V1R2M0,
//            DEV=3350,VOLSER=WORK01,
//            TAPE=TAPE,TVOLSER=FSI120,
//            TVRM=V1R2M0
//IEBCOPY EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//ASM      DD DSN=&PFX..&VRM..ASM,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=&DEV,VOL=SER=&VOLSER,
//            SPACE=(CYL,(1,0,36))
//*
//OBJ      DD DSN=&PFX..&VRM..OBJ,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=&DEV,VOL=SER=&VOLSER,
//            SPACE=(CYL,(1,0,36))
//*
//LOAD     DD DSN=&PFX..&VRM..LOAD,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=&DEV,VOL=SER=&VOLSER,
//            SPACE=(CYL,(1,0,36))
//*
//MLIB     DD DSN=&PFX..&VRM..MLIB,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=&DEV,VOL=SER=&VOLSER,
//            SPACE=(CYL,(1,0,36))
//*
//PLIB        DD DSN=&PFX..&VRM..PLIB,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=&DEV,VOL=SER=&VOLSER,
//            SPACE=(CYL,(1,0,36))
//*
//*
//ASMT        DD DSN=FSI.&TVRM..ASMT,
//            DISP=(OLD,KEEP,KEEP),
//            UNIT=&TAPE,
//            LABEL=(1,SL),
//            VOL=SER=FSI120
//*
//OBJT        DD DSN=FSI.&TVRM..OBJT,
```

```
//             DISP=(OLD,KEEP,KEEP),
//             UNIT=AFF=ASMT,
//             LABEL=(2,SL),
//             VOL=REF=*.ASMT
//*
//LOADT        DD DSN=FSI.&TVRM..LOADT,
//             DISP=(OLD,KEEP,KEEP),
//             UNIT=AFF=OBJT,
//             LABEL=(3,SL),
//             VOL=REF=*.OBJT
//*
//PLIBT        DD DSN=FSI.&TVRM..PLIBT,
//             DISP=(OLD,KEEP,KEEP),
//             UNIT=AFF=LOADT,
//             LABEL=(4,SL),
//             VOL=REF=*.LOADT
//*
//MLIBT        DD DSN=FSI.&TVRM..MLIBT,
//             DISP=(OLD,KEEP,KEEP),
//             UNIT=AFF=PLIBT,
//             LABEL=(5,SL),
//             VOL=REF=*.PLIBT
//*
// PEND
//*
//FSILOAD   EXEC FSILOAD
//SYSIN      DD *
   C O=ASM,I=ASMT
   C O=OBJ,I=OBJT
   C O=LOAD,I=LOADT
   C O=PLIB,I=PLIBT
   C O=MLIB,I=MLIBT
/*
//*
//
```

Download the JCL here: fsiload.jcl

Running FSI

Before you run FSI you must first allocate the MLIB and PLIB libraries to your TSO session. You can use the following commands:

```
alloc fi(fsimlib) da(fsi.v1r2m0.mlib) shr

alloc fi(fisplib) da(fsi.v1r2m0.plib) shr
```

Now you must give FSI access to the necessary load modules. You can do this two different ways. First you can add the FSI load library to your TSO Logon Procedure or you can allocate the FSI Load Library data set to the DDNAME FSILLIB.

```
alloc fi(fisllib) da(fsi.v1r2m0.load) shr
```

If you added the FSI LOAD Library dataset to your LOGON PROC you can invoke FSI directly as a TSO command:

READY
```
FSISTART SAMP01
```

If you use the FSILLIB method, you can use the TSO CALL command to invoke FSI.

```
READY
call 'fsi.v1r0m0.load(fsistart)' 'samp01'
```

FSISTART will set up the environment and then attach the program samp01 as a subtask. FSISTART then waits for the subtask to end.

Changing Your TSO Logon Proc

Using FSI

Changing Your TSO Logon Proc

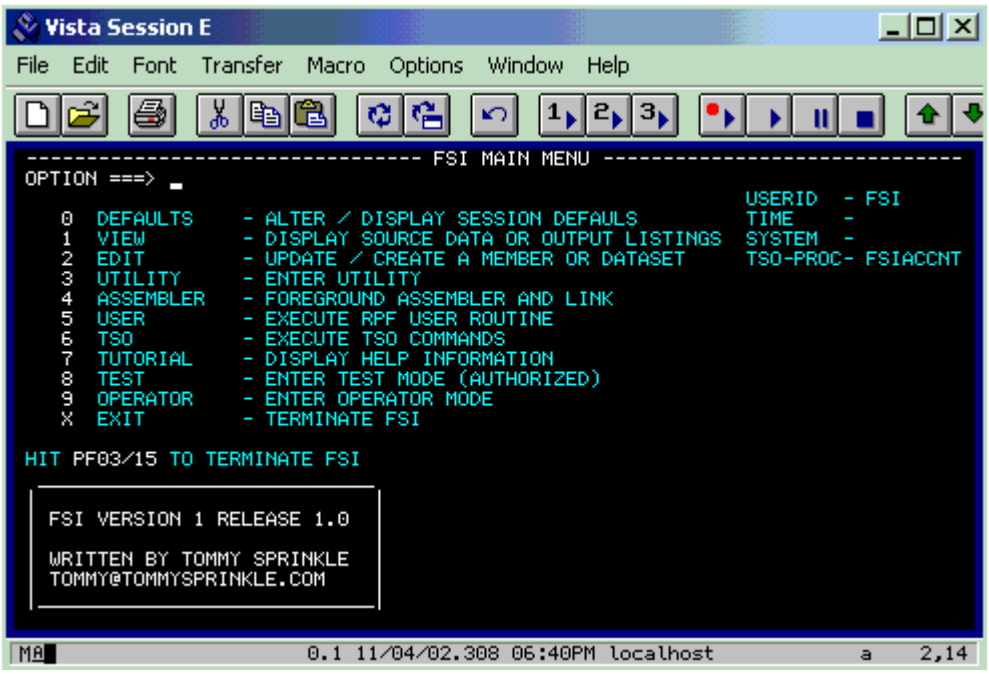
You can add FSI to your TSO Logon Proc if you desire. You will need to add three DD Statements to your logon proc.

You need to add a statement to your STEPLIB and DD Statements for the MLIB and PLIB. Use the example below to modify your Proc in SYS1.PROCLIB.

```
//IKJACCNT PROC
//IKJACCNT EXEC PGM=IKJEFT01
//STEPLIB DD DISP=SHR,DSN=RPF.V1R5M0.SRPFLOAD
//        DD DISP=SHR,DSN=FSI.V1R2M0.LOAD
//SYSHELP DD DISP=SHR,DSN=SYS1.HELP
//FSIPLIB DD DISP=SHR,DSN=FSI.V1R2M0.PLIB
//FSIMLIB DD DISP=SHR,DSN=FSI.V1R2M0.MLIB
//DD1     DD DYNAM
//DD2     DD DYNAM
//DD3     DD DYNAM
//DD4     DD DYNAM
//DD5     DD DYNAM
//DD6     DD DYNAM
//DD7     DD DYNAM
//DD8     DD DYNAM
//DD9     DD DYNAM
//DDA     DD DYNAM
//DDB     DD DYNAM
//DDC     DD DYNAM
//DDD     DD DYNAM
//DDE     DD DYNAM
```

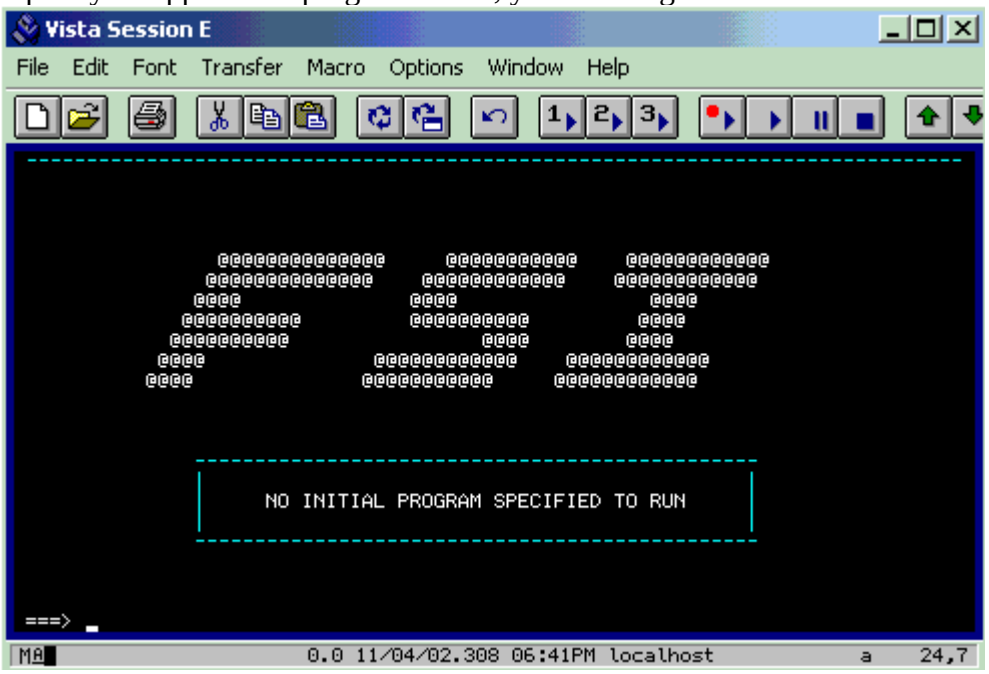
Once you have changed your TSO Logon Proc you need to logoff and logon again. You should then be able to invoke FSISTART as a command. You can try the second example program using:

fsistart samp02



You can see that samp02 displays a screen the very much looks like the initial RPF screen.

If you fail to specify an application program to run, you should get the default screen:



Using The API

Using the FSI API

The API

FSI Version 1, Release 2 implements the use of VDEFINE instead of the VGET and VPUT methods used with
the previous release. In addition the VDEFINE function there is also a DISPLAY function.

VDEFINE

VDEFINE is used to define program variables to FSI. A FSI variable is identified by a name consisting of one to eight characters. FSI variables can be a maximum length of 255 bytes long. The VDEFINE API function call specifies the variable name, the variable type (character or numeric), the data length and the data to set the variable to.

The $VDEFINE assembler macro is provided to invoke the VDEFINE function.

$VDEFINE(variable name , data address, variable type, data length, copy option)

Variable name is the name of the variable. It must be one to eight characters long. It can be specified as the address of an eight byte name (padded with blanks) or it can be specified as a constant.

Data address is the pointer to the variable in the program.

Variable type is a full word integer value indicating the data type. 1 = Numeric, 2 = Character

Data length is the length of the data string. It can be specified as the address of a full word containing the data length or it can be specified as a constant value.

The copy option is a full word integer value but is currently ignored.

Examples:
$VPUT('PF03',MYPF03,2,8,0)
This will set the value of the variable 'PF03' to an eight byte character string called MYPF03.

You can accomplish the same thing using the $VPUT macro without constants:
$VPUT(VNAME,VDATA,VLEN,VTYPE,0)

```
VNAME   DC   CL8'PF03'
VLEN    DC   F'3'
VDATA   DC   C'END'
VTYPE   DC   F'2'
```

To code the parameter list directly you would use:

```
PLST  DC    A(VNAME)
      DC    A(VDATA)
      DC    A(VTYPE)
      DC    A(VLEN)
      DC    X'80',AL3(VZERO)
```

```
VZERO  DC  F'0'
```

DISPLAY

DISPLAY is used to display a panel on the terminal and get a response from the terminal user. The DISPLAY function call specifies the name of the panel to display, an optional error message and an optional cursor position. If the error message is specified, the error message must be defined in the MLIB data set. If a cursor field is specified, the cursor will be placed into the screen field specified.

The $DISPLAY assembler macro is provided to invoke the DISPLAY function.

$DISPLAY(panel name , error message, cursor field)

Each of the parameters can either be specified as the name of an eight byte area (padded with blanks) or it can be specified as a constant.

```
$VDISPLAY('SAMP01','SAMP0101','USERID')
$VDISPLAY('SAMP01')

$VDISPLAY(PANEL,ERMSG,CSRFLD)

PANEL   DC  CL8'SAMP01'
ERMSG   DC  CL8'SAMP0101'
CSRFLD DC  CL8'USERID'
```

PANELS

Using FSI Panels

Using FSI Panels

FSI Panels are modeled after SPF panels. Only a limited subset of SPF functionality is currently implemented.

```
)ATTR DEFAULT(%+_)
  ! TYPE(OUTPUT) INTENS(HIGH)
  : TYPE(INPUT) INTENS(NON)
)BODY
+
+                                +TEST PANEL
+
+
+
+           %@@@@@@@@@@@@@@     @@@@@@@@@@@    @@@@@@@@@@@
+          %@@@@@@@@@@@@@@@    @@@@@@@@@@@    @@@@@@@@@@@
+         %@@@@                @@@@                @@@@
+        %@@@@@@@@@@          @@@@@@@@@@          @@@@
+       %@@@@@@@@@@                 @@@@         @@@@
+      %@@@@               @@@@@@@@@@@@     @@@@@@@@@@@@
+     %@@@@                @@@@@@@@@@@     @@@@@@@@@@@@
+
+ !Z +
+ +USERID%===>_Z +
+
+ +PASSWD%===>:Z +
+
+
+
+
+
+
%===>_ZCMD                                                              +
)PROC
 .ZVARS = '(ZSTAT USERID
                  PASSWD)'
)END
```

ATTR

The ATTR section specifies the field attributes. A minimum of three attributes must always be defined. By default these are "%", "+" and "_". The default attribute characters can be changed with the DEFAULT parameter on the)ATTR line.

The "%" attribute is defined as: TYPE(TEXT) INTENS(HI)

The "+" attribute is defined as: TYPE(TEXT) INTENS(LOW)

The "_" attribute is defines as: TYPE(INTPUT) INTENS(LOW)

Additional attribute characters can be defined in the ATTR section. You must specify the attribute

character followed by its characteristics:

TYPE(INPUT / OUTPUT / TEXT)

INTENS(HI / LOW)

CAPS(ON / OFF) *Not currently implemented

BODY

The BODY section defines the body of the panel. The attribute characters are used to define the beginning and ending of the various fields. An input or output variable is expected to be followed by a variable name. A special variable name of "Z" has a special meaning. The Z variable is a place holder and is replaced with a value specified in the PROC section. The Z variable can be used to assign a longer name to a short field.

The BODY section can contain a maximum of 24 lines.

PROC

The PROC section is provided so that ZVARS values can be specified. The ZVARS values are used in place of the place holder "Z" specified on the panel. The Z vars are replaced in the order they appear in the body from left to right, top to bottom.

If no ZVARS are used, the PROC section may be omitted.

END

The END section is used to signal the end of data for the panel definition.

Messages

Specifying Messages

Specifying Messages

FSI messages are defined in the MLIB data set. Each member in the MLIB can define one or more messages. The name of the member in the MLIB is the message ID minus the last character. For example MLIB member "SAMP010" will contain messages SAMP0101, SAMP0102, SAMP010A, etc.

```
SAMP0101 'INVALID USERID ENTERED'
SAMP0102 'HINT: USE ANYTHING FOR THE USERID'
SAMP0103 'INVALID PASSWORD ENTERED'
SAMP0104 'HINT: TRY USING ''SECRET'' FOR THE PASSWORD'
SAMP0105 'HINT: TRY USING ''END'' FOR A COMMAND'
```