MAP3270

Tutorial

Version 2.2.0

Issued October 2, 2021

A package to enable COBOL, PL/I(F) and Assembler programs to access 3270 terminals in full screen mode under TSO.

MAP3270 V2.2.0 – Software Package to implement full screen IO for IBM 3270 class terminals.

Table of Contents

Table of Figures

Introduction

Welcome to MAP3270. MAP3270 is a package to enable COBOL, PL/I(F) and Assembler programs to access 3270 terminals in full screen mode under TSO. It is assumed that the reader has some knowledge of using IBM3270 type terminals or emulators. It is not assumed that the reader knows how to format the data streams for fullscreen operation.

The idea for MAP3270 was first concieved when I wanted/needed to generate some TSO utility programs that ran in full screen mode without having ISPF available. ISPF is an IBM licensed product. ISPF allows panels to be defined using a text editor. ISPF panels are like drawings or markups of what the screen should look like. MAP3270 panels are very similar to ISPF panels (See Figure 1 on page 14).

How ISPF and MAP3270 panels are processed are different. Since ISPF is out of the scope here, it will not be discussed here any further.

The MAP3270 panels are compiled into physical and symbolic maps. The physical map is the template for the actual data stream that will be sent to the 3270. The symbolic map is to be included in your program. The MAP3270 software interfaces the physical and symbolic maps and requests the I/O with the 3270. Note that CICS has physical and symbolic maps. The concepts are the same but there is no compatability between CICS and MAP3270 maps.

Interfacing your program to the 3270 is pretty straight forward. Your program will fill in the data fields in the symbolic maps, call the interface routine, retrieve the data from the symbolic map and process the data. It might sound too simple but basically that all there is to it.

MAP3270 has been tested using a “as delivered” Hercules 3.12 unmodified Tur(n)key MVS and with the Vista 3270 emulator running under Windows 10. MAP3270 has also been tested using a “as delivered” Hercules 4.0 unmodified TK4- version 8 MVS and with the Vista 3270 emulator running under Windows 10. Please note that Vista refers to the 3270 emulator not to a version of Windows. Vista is stable and is the most accurate emulator I have used.

An additional component provided as part of MAP3270 is TTYRTN. It provide a line based interface similar to the way TSO acts when you run a program with a file assigned to DA(*). The biggest difference is you don’t keep getting the “***” at the bottom of each screen and the lines scroll up from the bottom not down from the top. TTYRTN was developed and implemented using MAP3270.

Version 2.1.0 introduced several new features and a couple of minor bug fixed. The fixes were originaly release as “hot patch 1”. New features included extended attribute support (i.e. color, highlighting, etc).

Version 2.2.0 introduces a new feature and some minor fixes. User defined field names are now limited to 28 characters vs. 6 in previous versions. The user can control what components are generated using the EXEC PARM=. Also, T3270IOP had an entry and an alias called T327IOP added so PL/I programs could do static calls.

Installing MAP3270

MAP3270 is pre-installed on distributions of TK4-. All of the executables are in 'SYS2.LINKLIB' all the other datasets have the high level qualifier 'MAP3270.' These instruction will create what I call a "test version". All the dataset are created with the high level qualifier 'userid.MAP3270'.

If you desire, the JCL can be modified to directly update the TK4- pre-installation. I recommend creating the "test" version and then copy the all the members to update the TK4- pre-installation.

1. Download the MAP3270-V2.2.0.ZIP archive/zip file. All of the .txt files are compile listings and/or installation listings.
2. Extract the MP3270.AWS tape and RESTORE.JCL from the archive.
3. Edit RESTORE.JCL if you may want to change the default userid HERC01, volume PUB002 and the tape drive for the tape (assumed to be 480). Submit the restore JCL. If you changed the userid, make sure you note it for the next steps. The following dataset will be created:

userid.MAP3270.ASM
Userid.MAP3270.CNTL
userid.MAP3270.COB
userid.MAP3270.MACLIB
Userid.MAP3270.MAP
userid.MAP3270.PANEL
userid.MAP3270.PLI
userid.MAP3270.SOURCE
userid.MAP3270.LOADLIB

Figure 1 - MAP3270 datasets

The restore JCL will delete backup copies of these dataset before renaming the existing datasets to userid.MAP3270.OLD.type. New empty datasets will be created by the restore JCL.

4. If the restore job executed with CC 0000 for all steps, you must now manually change additional members as indicated below. If you are running TK4-, these changes were probably made.

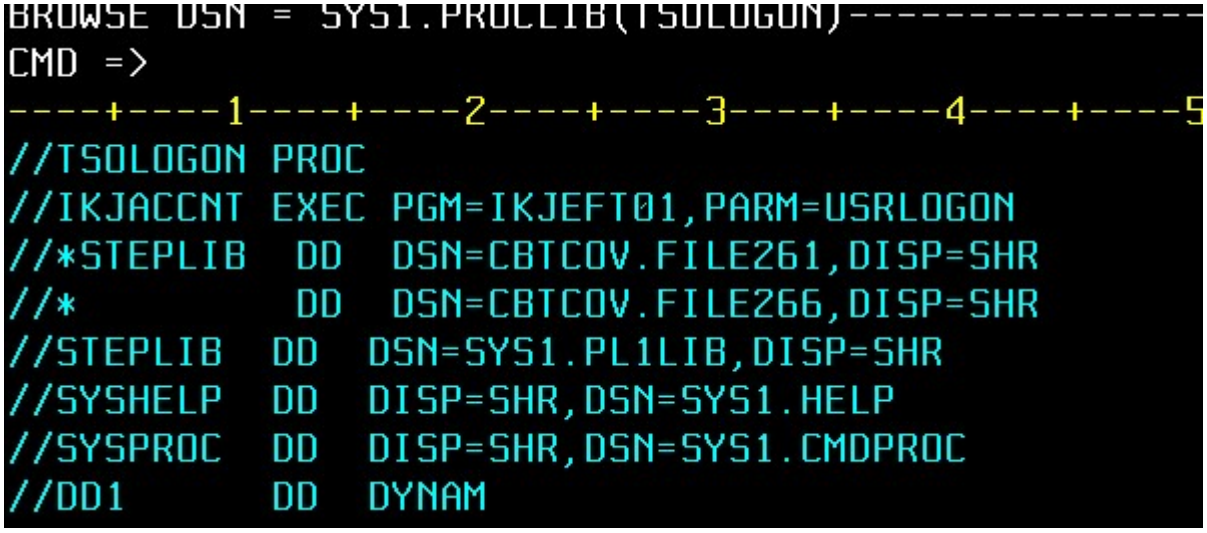

Figure 2 - TSO Logon Proc Sample

5. This step is required only if you intend to run PL/I(F) programs under TSO. Make sure that the STEPLIB includes the PL1LIB (see Figure 2 - TSO Logon Proc Sample on page 7) inting as indicated above. Note the PARM= since this must match a member name in SYS1.CMDPROC.
6. Note – there is a problem whenever you update the HERC01.CMDPROC. You may get a messages from your TSO session:

   OPERATOR ACTION HAS BEEN REQUESTED FOR YOUR DATA SET

   And on the system console, you will see:

   *TSU 2 *IEC507D E 191,MVSCAT,HERC01,IKJACCNT,HERC01.CMDPROC
   *TSU 2 *00 IEC507D REPLY 'U'-USE OR 'M'-UNLOAD

   This message indicate the file is expiration date protected. Reply “U” to allow the write. It may issue the message again. Reply “U’ again. The operation should be completed.

   It is possible to remove the expiration date but that is outside the scope of this document.

7. Set up the customized logon. If you do not have a HERC01.CMDPROC, you can create USRLOGON in SYS1.CMDPROC. Simply edit/create member name so it incorporates Figure 3 - Logon Command Proc. This name must match the name as indicated in the PARM= option in Figure 2 - TSO Logon Proc Sample.

```
BROWSE DSN = SYS1.CMDPROC(USRLOGON)----------------------Col 001,lin
CMD =>
----+----1----+----2----+----3----+----4----+----5----+----6----+----
        PROC 0
CONTROL NOMSG,NOLIST,NOSYMLIST,NOCONLIST,NOFLUSH
FREE FILE(SYSHELP)
WRITE Logging on to BSP1 at &SYSTIME using &SYSPROC
ALLOC FILE(SYSHELP) DSN('SYS1.HELP','SYS2.HELP') SHR
ALLOC FILE(X1) DSN('&SYSUID..CMDPROC(STDLOGON)') SHR
IF &LASTCC = 0 THEN +
   DO
      WRITE Logging on using private logon procedure
      FREE FILE(SYSPROC)
      FREE FILE(X1)
      ALLOC FILE(SYSPROC) DSN('&SYSUID..CMDPROC','SYS1.CMDPROC') SHR
   END
ELSE +
   DO
      WRITE Logging on using public logon procedure
      FREE FILE(X1)
   END
EXIT
```

Figure 3 - Logon Command Proc

8. If you do not have a HERC01.CMDPROC, you can create RUN3270 in SYS1.CMDPROC. This sets up the customized logon. If you do not have a HERC01.CMDPROC, you can create RUN3270 in SYS1.CMDPROC.

```
BROWSE DSN = HERC01.CMDPROC(RUN3270)------
CMD =>
----+----1----+----2----+----3----+----4--
PROC 1 PGM
CALL 'HERC01.MAP3270.LOADLIB(&PGM)'
```

Figure 4 - RUN3270 Clist

9. This should complete your manual portion. Logoff and logon to complete the install.
10. MAP3270 is ready to use now as is. Note – JCL is included to regenerate the MAP3270 system. It is not necessary to regenerate the package. Regeneration is optional.

11. This step is required only if you intend to run PL/I(F) programs under TSO. PL/I (F) requires that a SYSPRINT file be allocated. Any program compiled with PL/I(F) will simply return to READY if the SYSPRINT file is not allocated. Normally, you only need to do this once per TSO session. At the READY prompt, enter:
    - ALLOC F(SYSPRINT) DA(*) (recommended)
    - ALLOC F(SYSPRINT) DA(somedsn) ('NULLFILE' works but not advised)
12. At the READY propmpt, type the following. You will see something like Figure 7 - Sample Panel on page 9.

        RUN3270 ADEMO

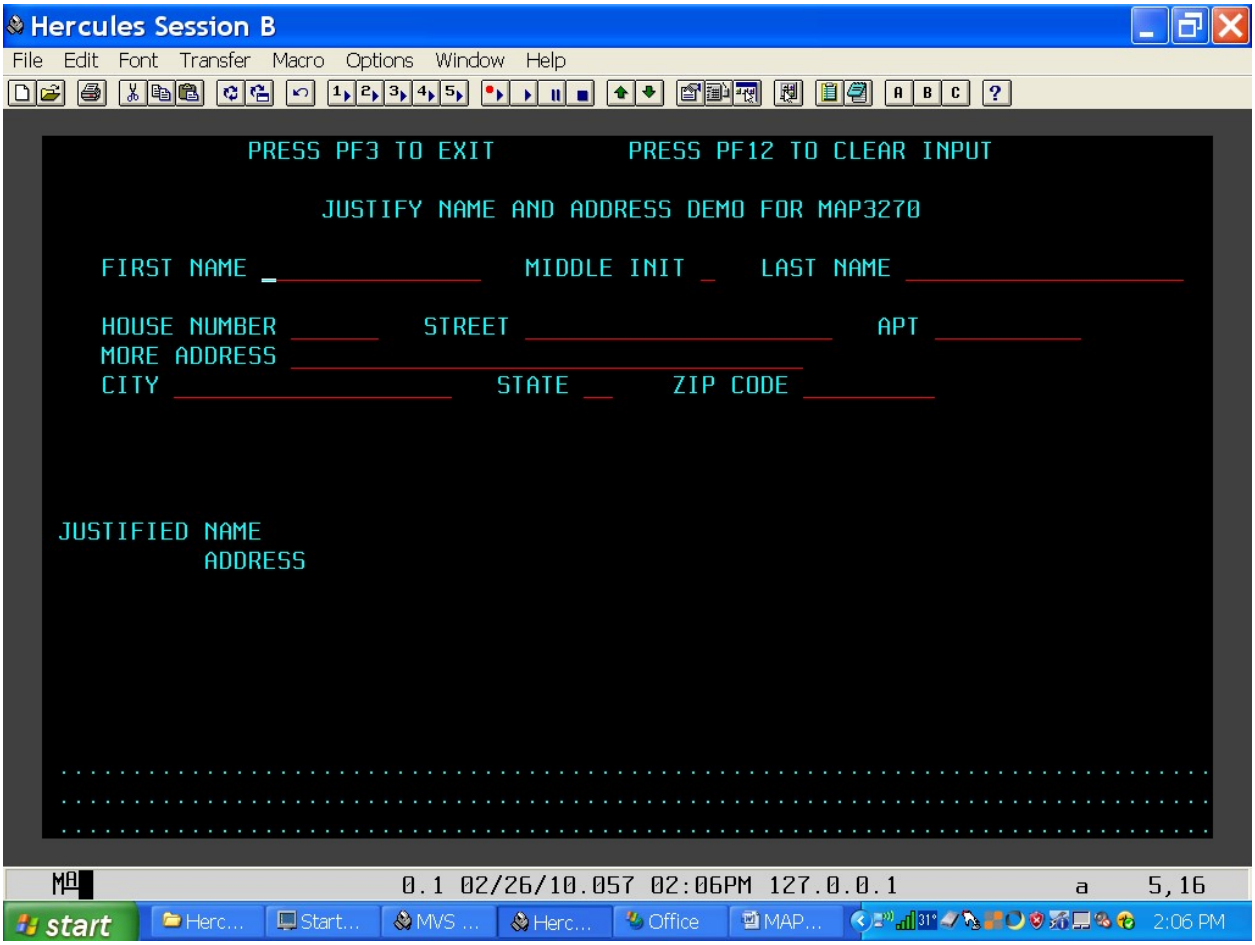

Figure 5 - Demo Screen

Sample Programs

There are sample programs in COBOL, Assembler and Pl/I(F) included. The source code can be found in userid.MAP3270.SOURCE. The names are ADEMO, CDEMO, MDEMO and PDEMO I suggest you try running at least one of then to verify that your install is complete and correct. MDEMO is an example of a program using multiple maps and extended attributes.

Dataset Naming Conventions

During installation, the following PDS datasets were created.

Userid.MAP3270.PANEL	This is suggested place to store your panels
Userid.MAP3270.MAP	This is where the assembler code for the physical map is generated.
Userid.MAP3270.MACLIB	This is where the macros used to assemble the map are stored.
Userid.MAP3270.LOADLIB	This is where the executable loadmodules are stored.
Userid.MAP3270.ASM	This is where the symbolic maps for assembler are generated into.
Userid.MAP3270.COB	This is where the symbolic maps for COBOL are generated.
Userid.MAP3270.PLI	This is where the symbolic maps for PL/I are generated.
Userid.MAP3270.CNTL	This is jcl to compile the entire MAP3270 package.
Userid.MAP3270.SOURCE	This is where the source code for the compiler and demo programs.

Figure 6 - MAP3270 default datasets

Creating a Panel

Panels are created using a TSO editor that support unnumbered files. A full screen editor such as FSE or RPF is the best to use. The dataset containing the panel should be fixed 80 bytes and unnumbered. It is recommended that you use a PDS.

Figure 7 - Sample Panel (see page 14) is a sample of what a panel definition looks like. This is the actual panel definition for the demo programs included in this package – ADEMO, CDEMO, and PDEMO (See page 10).

Note that a panel definiton consists of control statements and definition statement. If column 1 is a dot (.), the line is a control statement. Anything else is a definition statement.

A control statement starts with a dot (.) is column 1 followed by a key word. There are 4 controls statements the attr, name, screen, and vars.

.attr section

The .attr starts the definition of the 3270 attributes. Each attribute consists of a single character, except for a dot. This would indicate a control statement.

The syntax of an attribute definition is:

1. The attribute character in col 1
2. A series of traits seperated by commas starting in column 3. Traits are scanned from left to right. If there are conflicting traits are specified, the right most trait is what is used.
3. Comments starts after the second space.

The following is a list of traits.

Basic Traits

ALPHA	Field will contain alphanumeric data
NUM	Field will contain only numeric data
UNPROT	Field is unprotected – user can enter data in the field
PROT	Field is protected – user cannot enter data in the field
BRIGHT	Field is highlighted
NORMAL	Field is normal intensity
DIM	Field is dark
TITLE	This is field that is defined in the physical map only.

.

Supported Color Traits

This following lists all of the colors supported by MAP3270. Of course you 3270 emulator must support them for them to work.

DEFAULT	GREEN	BLACK	PGRN (pale green)
BLUE	TURQ (turquoise)	DBLUE (dark blue)	PTURQ (pale turquoise)
RED	YELLOW	ORANGE	GREY
PINK	WHITE	PURPLE	NEUTRAL

Supported HighlightingTraits

The following lists all off the supported highlighting options:

BLINK	REVERSE	ULINE (Underline)

The first attribute is being defined as an underscore. The next positional keywords define the trait as ALPHA and protected with normal intensity (by default). This means the alphanumeric data in the field cannot be changed by the user.

The next attribute is the % characted. It represents a numeric, unprotected, normal intensity field. This means only numeric data can be entered into the field. The remainder of the line is a comment.

All fields except those identified as TITLE are made available to the calling program. TITLEs are defined in the physical map but not in the symbolic map. In other words, the program cannot change a title.

.name section

The .NAME= allows you to specify what name should be used for the panel.

.screen section

The .screen start the definition of the actual fields to be on the panel. A field consists of an attribute followed by one or more characters and ends at next attribute character or end of line.

In Figure 1, the line immediately after the .screen defines 3 fields; the first is a series of blanks starting with the

1. @ in column 1 and extends to the charater before the next @,
2. the second is the text “PRESS PF3 TO EXIT” and extends to the next @
3. and the 3rd is PRESS PF12 T CLEAR THE INPUT and ends at the end of the line.

On the actual 3270 display, attributes are always displayed as a blank.

.var section

The .var defines how user defined varialbes are to be map to more meaning full names. By default, field names are generated in the format RxxCyy, where xx is the row and yy is the colum. The first Field is assigned the default name of R05C15 because it start on row 4, column 15. The var section allows you to assign a more symbolic name. In this example R05C15 is equivalent to FNAME. Now you can use FNAME in your program rather than R05C15. In V2.2.0, the names can be up to 28 characters. If you use the ‘-‘ or ‘_’ character in your names, they will be translated for the language being generated. For example, you define FIRST-NAME and you generate PL/I, the name would be changed to FIRST_NAME but kept as FIRST-NAME for COBOL.

Creating a Panel

The best way to create a panel is to use an editor to define a

1. mock up of what the screen should look like.
2. figure out what characteristics each field should have.
3. define attributes characters for each field

Using figure 1 as an example, the screen was mocked up using a text editor.

Now, go thru the screen and decide what you want the end user to do with a field. Some fields like “PRESS PF3 TO EXIT” and FIRST NAME are fixed and should not be changed. Fields like this are called titles. In our example, in the attr section, we defined a the “@” as a TITLE. Titles only display on the screen and are not accessible to the program. Now we can go thru the mock up and put the @ in front of the first character of each title.

```
BROWSE DSN = HERC01.MAP3270.PANEL(DEMOPAN)-----------------Col 001,line 0000001
CMD => _
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----
.ATTR
_ ALPHA,PROT
% NUM,UNPROT SO IT THIS
: ALPHA,UNPROT,BRIGHT
$ NUM,PROT THIS IS A COMMENT
@ TITLE
.NAME=DEMOPAN
.SCREEN
@            @PRESS PF3 TO EXIT        @PRESS PF12 TO CLEAR INPUT
@
@                 @JUSTIFY NAME AND ADDRESS DEMO FOR MAP3270
@
@  @FIRST NAME:..............@ @MIDDLE INIT:.@ @LAST NAME:...................@
@
@  @HOUSE NUMBER:......@ @STREET:.....................@ @APT:.........@
@  @MORE ADDRESS:.....................................@
@  @CITY:...................@ @STATE:..@  @ZIP CODE:.........@
@
@
@
@
BROWSE DSN = HERC01.MAP3270.PANEL(DEMOPAN)-----------------Col 001,line 0000022
CMD => _
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----
@JUSTIFIED NAME    _.............................................
@         @ADDRESS _.............................................
@                  _.............................................
@                  _.............................................
@                  _.........
@
@
@
_.............................................................................
_.............................................................................
_.............................................................................
.VARS
R05C15=FNAME
R05C45=MNAME
R05C59=LNAME
```

Figure 7 - Sample Panel

Compiling A Panel

Introduction

Once the panel "drawing" is complete, it is time to compile it into physical and symbolic maps. The MAP3270 compiler reads in the panel "source" code and produces a physical map, which is an assembler equivalent, and an assembler, Cobol, and PL/I symbolic maps. The symbolic map is the code you insert in you program. This is the area that passes data to and from the screen.

The physical map is assembler code which you compile and link edit like any other assembler program. The symbolic maps are copied or included into you application program.

Running the Panel Compiler

A proc is supplied for running MAP3270. Figure 8 - Proc to compile a panel – can be used as an inline proc or can be copied to your proclib (SYS1.PROCLIB or SYS2.PROCLIB).

```
BROWSE DSN = HERC01.CLE.DISTRO.CNTL(MAP3270P)--------------Col 001,lin
CMD =>
----+----1----+----2----+----3----+----4----+----5----+----6----+----7-
//MAP3270 PROC PAN=,
//             LIB='HERC01.MAP3270.LOADLIB',
//             SOUT='*',
//             QUAL='HERC01.MAP3270'
//MAP3270 EXEC PGM=MAP3270,REGION=1024K
//STEPLIB  DD DSN=&LIB,DISP=SHR
//         DD DSN=SYS1.PL1LIB,DISP=SHR
//SYSPRINT DD SYSOUT=&SOUT
//MAP      DD DSN=&QUAL..MAP(&PAN),DISP=SHR
//ASM      DD DSN=&QUAL..ASM(&PAN),DISP=SHR
//COB      DD DSN=&QUAL..COB(&PAN),DISP=SHR
//PL1      DD DSN=&QUAL..PLI(&PAN),DISP=SHR
//SYSIN    DD DSN=&QUAL..PANEL(&PAN),DISP=SHR
//*
//ASM      EXEC PGM=IFOX00,PARM='OBJ,NODECK',REGION=128K,
//             COND=(16,LT,MAP3270)
//SYSLIB   DD   DSN=&QUAL..MACLIB,DISP=SHR
//SYSUT1   DD   DSN=&&SYSUT1,UNIT=SYSDA,SPACE=(1700,(600,100))
```

Figure 8 - Proc to compile a panel

To compile a panel, use the following JCL:

```
//HERC01M JOB MSGLEVEL=(1,1),CLASS=A,MSGCLASS=A,NOTIFY=HERC01
//S1 EXEC MAP3270,PAN=DEMO2PA
```

This jcl will compile a panel called DEMO2PA into the standard map datasets as well as generate the physical map. In V2.2.0, the assembler, COBOL and PL/I items as well as the physical map will be generated by default. To specify which components you want generated, use the EXEC

//S1 EXEC MAP3270,PAN=DEMO2PA,PARM.MAP3270='ASM,COB,PLI,MAP'

The valid items are ASM, COB, PLI, MAP, NOASM, NOCOB, NOPLI, and NOMAP. They can be listed in any order as long they are separated by commas.

By default, the listing produced by the assembler step will generate everything (i.e. no PRINT op).. If the assembler you are using supports SYSPARM and you want a "PRINT" statement (i.e. PRINT NOGEN) included, use this JCL:

```
//HERC01M JOB MSGLEVEL=(1,1),CLASS=A,MSGCLASS=A,NOTIFY=HERC01
//S1 EXEC MAP3270,PAN=DEMO2PA,PARM.ASM='OBJ,NODECK,SYSPARM=NOGEN'
```

The SYSPARM will cause the TSECT macro to generate a "PRINT NOGEN" statement.

Physical Map

The physical map is assembler code generated into the .MAP dataset.

Symbolic Map

```
BROWSE DSN = HERC01.MAP3270.COB(DEMOPAN)-------------------Col 001,line 0000001
CMD => _
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----
      ** GENERATED BY MAP3270 V2.0.0 ON 100108 AT 052704700             0000010
       01  DEMOPAN-PARM.                                               0000020
           05  DEMOPAN-NAME            PIC X(8) VALUE 'DEMOPAN'.       0000030
           05  FILLER                  PIC S9(8) COMP VALUE ZERO.      0000040
           05  DEMOPAN-RET             PIC S9(4) COMP.                 0000050
           05  DEMOPAN-AID             PIC X.                          0000060
           05  DEMOPAN-SBA             PIC XX.                         0000070
           05  FILLER                  PIC X.                          0000080
           05  FNAME.                                                  0000090
               10  FNAME-C             PIC X(01) VALUE ' '.            0000100
               10  FNAME-L             PIC S9(4) VALUE +15 COMP.       0000110
               10  FNAME-D             PIC X(15) VALUE                 0000120
               '...............'.                                      0000130
           05  MNAME.                                                  0000140
               10  MNAME-C             PIC X(01) VALUE ' '.            0000150
               10  MNAME-L             PIC S9(4) VALUE  +1 COMP.       0000160
               10  MNAME-D             PIC X(01) VALUE '.'.            0000170
           05  LNAME.                                                  0000180
               10  LNAME-C             PIC X(01) VALUE ' '.            0000190
               10  LNAME-L             PIC S9(4) VALUE +19 COMP.       0000200
               10  LNAME-D             PIC X(19) VALUE                 0000210
```

Figure 9 - COBOL Symbolic Map

The MAP3270 system supports Assembler, COBOL and PL/I(F) programs. I will use COBOL as the demo but the process is the same for all.

First, you must use a COPY statement (INCLUDE for PL/I) to define the map to you program as well as copy the predefined AIDCODES. Using figure 2 as a guide, the name DEMOPAN-NAME has the value DEMOPAN. This is the name specified in the .NAME= field.

DEMOPAN-RET is the RETURN-CODE derived by the interface program.

DEMOPAN-AID is the AID (attention id key code). This is a one character code that represents the key that started the action (i.e. Enter, PFK01, etc.). All of the possible AID codes are defined in the copy member AIDCODES.

DEMOPAN-SBA is the address of the cursor when the AID key is pressed. Each character positions on the screen has a unique address. This field contains the row and column of the cursor. If you want code to code/decode the SBA address, Jay Moseley has one available for download at: http://www.jaymoseley.com/hercules/miscpgms.htm#Translate3270BufferAddresses

Now the layout repeats. Each of the items represents a field = FNAME, MNAME, LNAME.

For writing to the screen:

1. The –C field is used to indicate where the cursor is to be placed when a write takes place. The first –C field with a value of X (scanning from top to bottom) gets the cursor.
2. The –L indicates the length of the field. Nomally you do not change this value.
3. The –D contains the actual data to be sent to the screen.

After reading from the screen:

1. The –C field is set to space.
2. The –L indicates the length of the field. Nomally this value is not changed.
3. The –D contains the actual data that was entered into the screen.

Extended Attributes

Introduction

The addition of extended attributes allows the use of colors and highlighting. All 16 colors are supported as well as underlining, blinking and reverse video. No modifications to existing panels need to be made.

Compatability Note

The 3270 manuals describe the standard way to format the datastream. There is an incompatibility with MVS 3.8j that results in data being interpreted as attributes. For this reason, MAP3270 has the option of generating standard vs MVS 3.8j compatable maps. Since this package is intended for the Tur(n)key 3.8j system, the default is MVS 3.8j. There are two ways to generate “standard” 3270 datastreams, One is to change the default value on the TSTPAN macro. The other is to add the standard option to the TSTPAN in your .MAP.

See the “.attr section” on page 11 for a list of all the traits.

Changing Map Generation Default

To change the default code generation mode, simpley edit the TSTPAN macro (see Figure 10 - TSTPAN Macro on page 19).

```
BROWSE DSN = HERC01.MAP3270.MACLIB(TSTPAN)-----------------Col 001,lin
CMD => _
----+----1----+----2----+----3----+----4----+----5----+----6----+----7-
         MACRO
&NAME    TSTPAN &OSVER=B
.*************************************************************************
.*
.*   TSTPAN - START THE DEFINITION OF A 3270 PANEL.  GLOBAL OPTIONS
.*   ARE SPECIFIED HERE AS WELL.
.*   CURRENT GLOBAL OPTIONS:
.*      &ZOSVER  THIS CONTAINS THE VERSION MVS THE MAP SHOULD BE
.*               GENERATED FOR.  SOME VERSIONS OF MVS DO NOT SUPPORT
.*               THE EXTENDED ATTRIBUTES CORRECTLY.
.*               THIS VALUE IS CODED
.*               A = STANDARD VERSION PER 3270 MANUALS.
.*               B = MVS 3.8J EXTENDED ATTRIBUTES ARE NOT SUPPORTED
.*                   PER 3270 MANUALS.
.*
.* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Figure 10 - TSTPAN Macro

To change the default generation code, edit the TSTPAN macro and the string “&OSVER=B” to “&OSVER=A”.

Calling the MAP3270 Interface

Introduction

Using the MAP3270 interface is basically the same for all three suported languages. A parameter block is created (called an I/O Control Block) and a call to the MAP3270 interface is made. The default name created during installation is T3270IO. If you going to use PL/I, the default name is T327IOP.

In version 2.2.0, after each call to the interface, the variable IOCBRC should be tested for a non-zero value. A non-zero IOCBRC indicates a problem was encountered and allows the application to handle it. Previous versions displayed any problems on the system console and abended with code U3270.

First, an ‘open’ call must be made to the interface T3270IO. This is done by setting the IOCBFUNC to ‘STRT’ and calling T327IO. This initializes the MAP3270 interface.

Next, an ‘IO ’ call must be made to the interface T3270IO to perform an I/O. Actualy it is an Output/Input call.

When the program is ending, a ‘END ’ call must be made to the interface T3270IO.

T3270IO takes the data from the symbolic map and moves it into the physical map. The physical map is sent to the 3270. Upon return, the data stream received from the 3270 is mapped into the symbolic map and control returns to the caller.

Static vs Dynamic Calls

The interface may be called via static or dynamic calls. A static call results in the interface being included in each executable program. A dynamic call loads the interface into memory at run time. There is a utility in the distribution package called DYNALOAD which documents dynamic calls. I recommend using dynamic calls rather than static calls. All of the calls in the demo programs are dynamic calls. In Version 2.2.0, for PL/I programs, you can use T3270IOP or T327IOP for dynamic calls.

Sample Programs

A annotated compile listing is available for a sample COBOL, PL/I, and assembler program. Using these listing along with this document should help you to understand how to use MAP3270.

COBOL

These is the model for the COBOL IOCB and calls.

```
01  IOCB.
    05  IOCBFUNC                PIC X(4).
        88  IOCBFUNC-STRT            VALUE 'STRT'.
        88  IOCBFUNC-END             VALUE 'END '.
        88  IOCBFUNC-IO              VALUE 'IO  '.
    05  IOCBRC                  PIC S9(4) COMP.
    05  IOCBLEN                 PIC S9(4) COMP.
    05  IOCBADDR                PIC S9(8) COMP.
    05  FILLER                  PIC X(20).

Static Call:

    CALL 'T3270IO' USING IOCB  01-for-symbolic-map.

Dynamic Call:

      CALL 'DYNALOAD' USING dyna-parm  IOCB  01-for-symbolic-map.
```

Assembler

This is the model for Assembler IOCB.

```
IOCB     DS    0F
IOCBFUNC DS    CL4                 FUNCTION CODE (STRT/IO/END)
IOCBRC   DS    AL2                 RETURN CODE
IOCBLEN  DS    AL2                 IO BUFFER LENGTH
IOCBADR  DS    A                   IO BUFFER ADDRESS
```

```
          DS    XL20                    RESERVED
IOCBLN    EQU   *-IOCB
```

PL/I(F)

This is the model for PL/I.

```
DECLARE
     (IOCBFUNC_STRT            INIT('STRT'),
      IOCBFUNC_END             INIT('END '),
      IOCBFUNC_IO              INIT('IO  '),
      IOCBFUND_PUT             INIT('PUT ')) STATIC CHAR(4),
    1  IOCB,
      5  IOCBFUNC               CHAR(4)        INIT((4)' '),
      5  IOCBRC                 FIXED BIN(15)  INIT(0),
      5  IOCBLEN                FIXED BIN(15)  INIT(0),
      5  IOCBADDR               FIXED BIN(31)  INIT(0),
      5  FILLER                 CHAR(20);
```

IOCBRC Values

After each call to the interface, the return code should be checked for a non-zero value. Below is a list of the possible return codes in IOCBRC. These apply to all languages.

Return Cd	Description
0	Normal return – no problems detected
4	Invalid value in IOCBFUNC
8	When IOCBFUN C is STRT, IOCBADDR must be zero.
12	When IOCBFUNC is END, IOCBADDR must not be zero
16	The expected lengths of the symbolic and physical maps differ
20	An invalid or unexpected message came from the terminal
24	A field differs between the symbolic and physical maps.
28	An unexpected field was found. Probably the symbolic and physical maps differ.
32	A field length differs between the symbolic and physical maps.
1xx	TPUT failed. XX can be 00 through 44 (decimal). See appendix 1 page 27.
2xx	TGET failed. XX can be 00 through 28 (decimal) . See appendix 2 page 28.
300	LOAD failed. S806 trapped. Most likely the physical map is not linked edit correctly.

Figure 11 - List of IOCBRC return codes

Calling TTYRTN

Introduction

An additional component provided as part of MAP3270 is TTYRTN. It provides a line based interface similar to the way TSO acts when you run a program with a file assigned to DA(*). The biggest difference

is you don't keep getting the "***" at the bottom of each screen and the lines scroll up from the bottom not down from the top. If you are familiar with VM/CMS, this will look familiar.

TTYRTN is an assembler subroutine developed using MAP3270. TTYRTNP is the version for use with PL/I(F).

```
BROWSE DSN = USER1.MAP3270.COB(TTYPARM)--------------------Col 001,line 0000001
CMD =>
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----
000100*************************************************************** 0000000
000200*                                                               * 0000000
000300*   TTYPARM - PARM AREA FOR TTY3270.                            * 0000000
000400*                                                               * 0000000
000500*************************************************************** 0000000
000600 01  TTY-PARM.                                                   0000000
000700     05  TTY-FUNC-CD          PIC X(4).                          0000000
000800         88  TTY-FUNC-STRT        VALUE 'STRT'.                  0000000
000900         88  TTY-FUNC-END         VALUE 'END '.                  0000000
001000         88  TTY-FUNC-CLR         VALUE 'CLR '.                  0000001
001100         88  TTY-FUNC-PUT         VALUE 'PUT '.                  0000001
001200         88  TTY-FUNC-IO          VALUE 'IO  '.                  0000001
001300         88  TTY-FUNC-PUT-GET     VALUE 'PG  '.                  0000001
001400     05  TTY-LINE-OUT         PIC X(79).                         0000001
001500     05  TTY-LINE-IN          PIC X(78).                         0000001
001600     05  TTY-ERR-MSG          PIC X(79).                         0000001
001700     05  TTY-AID-CD           PIC X.                             0000001
```

Figure 12 - COBOL Interface to TTYRTN

There is an equivalent for PL/I. This interface is basically the same for all three suported languages. A parameter block is defined (called a TTY-PARM).

Three demo programs, TTYASM, TTYCOB and TTYPLI are provided.

If you run the TTYCOB program, you will see a screen similar to Figure 13 - Sample TTYCOB screen on page 23. See Figure 14 - Sample COBOL Code (page 25) that would produce the sample screen.

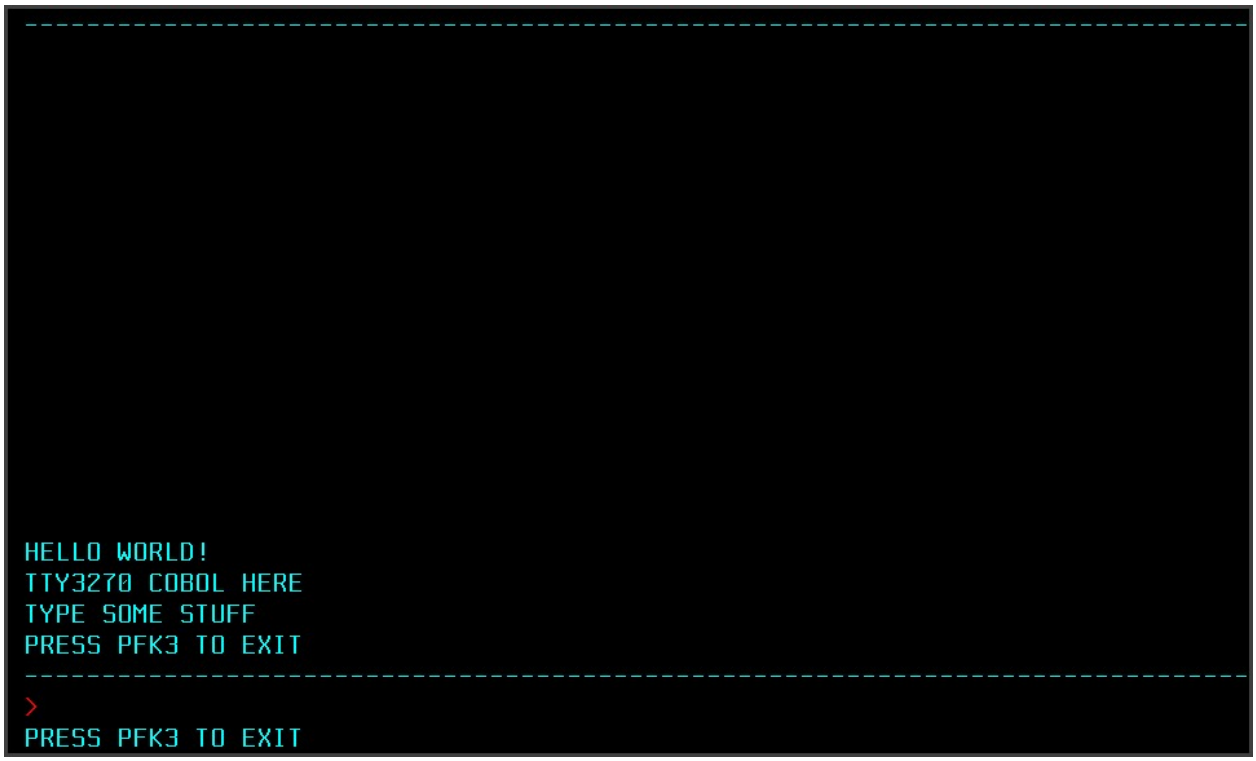

Figure 13 - Sample TTYCOB screen

The only area that you can type in is line 23 - the line with the ">" character.

First, an 'open' call must be made to the interface TTYRTN. This is done by setting theTTY-FUNC-CD to 'STRT' and calling TTYRTN. This initializes the TTYRTN and the MAP3270 interface.

Each time a line is added to the screen, the top most line scrolls up and off the screen and the new line is displayed on line 22.

To display a line without getting input, a 'PUT' function should be used. In this sample screen, the lines "Hello World", TTY3270 COBOL HERE" and "TYPE SOME STUFF" were displayed with the PUT function. For a PUT, the

- TTY-LINE-OUT should contain the line to display on line 22.
- TTY-LINE-IN should be blank
- TTY-ERR-MSG should be blank.

See the source code for TTYCOB for more info.

Next, an 'IO ' call must be made to the interface TTY3270 to perform an I/O. Note the PUT-GET is the same as an IO. For a PG, the

- TTY-LINE-OUT should contain the line to display on line 22.
- TTY-LINE-IN should be blank or a default response
- TTY-ERR-MSG should be blank or contain a message to display on line 24.

When the program is ending, a 'END ' call must be made to the interface TTYRTN.

Static vs Dynamic Calls

The TTYRTN interface may be called via static or dynamic calls. A static call results in the interface being included in each executable program. A dynamic call loads the interface into memory at run time. There is a utility in the distribution package called DYNALOAD which documents dynamic calls. I recommend using dynamic calls rather than static calls. All of the calls in the demo programs are dynamic calls.

Sample COBOL Program

Complete source code can be found in the userid.MAP3270.SOURCE(TTYCOB).

In working storage, COPY TTYPARM. See Figure 12 - COBOL Interface to TTYRTN on page 22.

```
Static Call:

    CALL 'TTYRTN' USING TTY-PARM.

Dynamic Call:

     CALL 'DYNALOAD' USING dyna-parm  TTY-PARM.
```

Figure 14 - Sample COBOL Code

```
PROCEDURE DIVISION.
     MOVE 'STRT'  TO TTY-FUNC-CD.
 TEST-AGAIN.
     PERFORM 9999-DYNA-CALL-1.
*
     MOVE 'PUT '               TO TTY-FUNC-CD.
     MOVE 'HELLO WORLD!'       TO TTY-LINE-OUT.
     MOVE SPACES               TO TTY-LINE-IN.
     MOVE SPACES               TO TTY-ERR-MSG.
     PERFORM 9999-DYNA-CALL-1.
*
     MOVE 'PUT '               TO TTY-FUNC-CD.
     MOVE 'TTY3270 COBOL HERE' TO TTY-LINE-OUT.
     MOVE SPACES               TO TTY-LINE-IN.
     MOVE SPACES               TO TTY-ERR-MSG.
     PERFORM 9999-DYNA-CALL-1.
*
     MOVE 'PUT '               TO TTY-FUNC-CD.
     MOVE 'TYPE SOME STUFF   ' TO TTY-LINE-OUT.
     MOVE SPACES               TO TTY-LINE-IN.
     MOVE SPACES               TO TTY-ERR-MSG.
     PERFORM 9999-DYNA-CALL-1.
*
     MOVE 'PRESS PFK3 TO EXIT' TO TTY-LINE-IN.
     MOVE SPACES               TO TTY-LINE-OUT.
     MOVE 'PRESS PFK3 TO EXIT' TO TTY-ERR-MSG.
     PERFORM 1000-ECHO-LINE UNTIL TTY-AID-CD = AIDPFK03.
*
     MOVE 'PG  '               TO TTY-FUNC-CD.
     MOVE 'YES'                TO TTY-LINE-IN.
     MOVE 'ARE YOU SURE?'      TO TTY-LINE-OUT.
     MOVE 'PRESS PFK3 TO EXIT' TO TTY-ERR-MSG.
     PERFORM 9999-DYNA-CALL-1.
     IF TTY-AID-CD = AIDPFK03 AND TTY-LINE-IN = 'YES'
         NEXT SENTENCE
     ELSE
         MOVE 'CLR '           TO TTY-FUNC-CD
         GO TO TEST-AGAIN.
*
     MOVE 'END '  TO TTY-FUNC-CD.
     PERFORM 9999-DYNA-CALL-1.
```

Trouble Shooting

This section contain a summary of notes and other suggestions extracted from the support group.

PL1LFCL proc not found.

I have not found the root cause of this. The initial solution to this is to copy the PL1LFCLG proc and delete the GO step.

Run3270 abends with S806

System completion code S806 means “module not found”. Make sure the load lib you are calling your 3270 program from has the following modules:

- TTYRTN
- TTY3270
- T3270IO

If you use PL/I, the following must be present in addition to the above:

- TTYRTNP
- T3270IOP

UpperCase A and B disappear

If the first character in a data fields is an upper case A or B, it and the following character are “disappearing”, MAP3270 V2.1.0 fixes this problem. This is an example of the incompatability of 3270 datastreams and MVS 3.8j. Simply regenerate the map and the problem should be resolved.

Appendix 1 – Return Codes From TPUT

Return code dec(Hex)	Meaning
0(0)	TPUT completed successfully.
4(4)	NOWAIT was specified and no terminal output buffer was available.
8(8)	An attention interruption occurred while TPUT was processing. The message was not sent.
12(C)	A TPUT macro instruction with an ASID operand was issued but the user, indicated by the ASID, requested that interterminal messages not be printed on the terminal. The message was not sent.
16(10)	Incorrect parameters were passed to TPUT.
20(14)	The terminal was logged off and could not be reached. Cross-memory TPUT could not get buffer or a serious error has occurred in z/OSMF ISPF.
24(18)	The sender is not permitted to send a message to the intended user.
28(1C)	The intended receiver of the message is logged on at a security label too low to receive the message.
32(20)	No storage is available.
36(24)	JESXCF at remote side is downlevel.
40(28)	JESXCF at local side is downlevel.
44(32)	JESXCF function call failed.

Appendix 2 - Return codes from TGET

Return code dec(Hex)	Meaning
0(0)	TGET completed successfully. Register 1 contains the length of the input line read into your input buffer.
4(4)	NOWAIT was specified and no input was available to be read into your input buffer.
8(8)	An attention interruption occurred while TGET was processing. The message was not received.
12(C)	Your input buffer was not large enough to accept the entire line of input entered at the terminal. Subsequent TGET macro instructions will obtain the rest of the input line.
16(10)	Incorrect parameters were passed to TGET.
20(14)	The terminal was logged off and could not be reached or a serious error has occurred in z/OSMF ISPF.
24(18)	TGET completed successfully. Register 1 contains the length of the input line read into your buffer. The data was received in NOEDIT mode.
28(1C)	Your input buffer was not large enough to accept the entire line of input entered at the terminal. Subsequent TGET macro instructions will obtain the rest of the input line. The data was received in NOEDIT mode.