SYSD/JFT®

Job and File Tailoring for CPMS®/SYSD®

Release 6.4.2

Reference Manual

Published January 1999

This edition, CUR-JFT0642, applies to release 6.4.2 of H&W Computer System's SYSD/JFT (Job and File Tailoring) option for CPMS/SYSD and to all subsequent releases and modifications until otherwise indicated in new editions or technical manual updates.

Order No. CUR-JFT0642

Comments A reader's comment form is provided at the back of this manual. If the form has been removed, send comments to H&W Computer Systems, Inc. at:

P.O. Box 46019 Boise, ID 83711	12438 W. Bridger Street, Suite 100 Boise, ID 83713
Main: (208) 377-0336	Customer Support: (208) 377-8436
Fax: (208) 377-0069	
World Wide Web: http://www.hwcs.com	E-mail: support@hwcs.com

Contents

About this Manual . . . . *vii*
Manual Organization viii
Conventions ix
  Text and Keyboard Conventions ix
  Symbol Conventions ix
Related Publications x

Chapter 1 *Introduction* *1*
Components of SYSD/JFT 2
  Panels 2
  Skeletons 2
  Messages 3
Storing Panels, Skeletons, and Messages 4
JFT Variables 5
Executing JFT Panels 6

Chapter 2 *Panel Sections* *7*
)ATTR Section 8
  Default Attribute Characters 9
  Overriding the Default Attribute Characters 9
)BODY Section 10
)INIT Section 12
)PROC Section 14
)END Section 17

Chapter 3 *Variables* *19*
System Variables 20
  &Z 20
  &ZCMD 20
  &ZCSR 20
  &ZDATE 20

&ZJDATEST 20
&ZDAY 20
&ZJDATE 20
&ZJ4DATE 21
&ZMONTH 21
&ZSEL 21
&ZTIME 21
&ZSTDYEAR 21
&ZYEAR 21
Profile Variables 22
&$DEPART 22
&$DESTID 22
&$NAME 22
&$TITLE 22
&$USER 22
Control Variables 23
.CURSOR 23
.CURSORF 23
.MSG 23
.MSGF 24
.SYSDTTL 24
.TRAIL 24
User Variables 25

Chapter 4 ***Logic Statements 27***
Syntax Rules 28
Assignment Statement 29
Concatenation Rules 29
Examples 30
FILESKEL Statement 32
IF Statement 34
LINK Statement 36
SUBSKEL Statement 37
TRANS Statement 38
TRUNC Statement 40
VER Statement 42
VER ALPHA 42
VER BIT 43
VER DSNAME 43
VER HEX 43
VER LIST 43
VER NAME 43
VER NB 44
VER NUM 44
VER PICT 44
VER RANGE 45

Chapter 5 **Skeleton and Message Files** 47

Skeleton Files 48

Control Statements 48

)BLANK 48

)CM 48

)DEFAULT 49

)IM 49

)TB 49

Message Files 52

Chapter 6 **Calling CICS Programs** 53

Communicating Between Panels and CICS Programs 54

$CMVGET 54

$CMVPUT 55

Calling CICS Programs from SYSD/JFT 56

Calling the JFTADD Program 56

SYSD-like Programs 58

Storing Data in JFT-like Programs 58

Constant Data 58

Local Variables 58

Global Variables 59

JFTADD Program 59

Appendix A **CPMS/SYSD Menu System Variables** 61

Appendix B **Sample Panels** 69

Primary Menu 70

System Utilities Panel 72

Submit IEBCOPY Panel 73

Appendix C **JFTADD Program** 77

Index 81

About this Manual

The *SYSD/JFT Reference Manual* explains how to create panels users can access through SYSD or CPMS. This manual is used by the programmer responsible for creating and maintaining the JFT panels.

Manual Organization

The *SYSD/JFT Reference Manual* is organized as follows.

Chapter 1, Introduction

Briefly describes SYSD/JFT, including JFT's components; storing panels, skeletons, and messages; JFT's variables; and executing JFT panels.

Chapter 2, Panel Sections

Describes the sections a panel can have and the parameters for each one.

Chapter 3, Variables

Describes the system, profile, control, and user variables you can use in the)INIT,)PROC, and)BODY sections.

Chapter 4, Logic Statements

Describes the statements you can use in the)INIT and)PROC sections.

Chapter 5, Skeleton and Message Files

Explains how to use skeleton and message files.

Chapter 6, Calling CICS Programs

Explains some of the logic behind the design of the JFT panels, including communicating between panels and CICS programs.

Appendix A, CPMS/SYSD Menu System Variables

Lists the variables available to JFT from the CPMS/SYSD menu system.

Appendix B, Sample Panels

Provides samples of some SYSD/JFT panels and the code that generates them.

Appendix C, JFTADD Program

Provides the complete code for the JFTADD program.

Conventions

The *SYSD/JFT Reference Manual* uses the following conventions.

Text and Keyboard Conventions

This kind of text	*Identifies*
BOLD bold	Commands and text you type. Uppercase bold text represents information you must type exactly as it appears. Lowercase text represents information you must substitute with the appropriate text. For example, when you see **variable_name** in the syntax of a logic statement, type the appropriate variable name.
italic	Field names, manual titles, and system messages. It is also used to introduce new words.
Enter	Special keys on the keyboard you press. The example here represents the Enter key.

Symbol Conventions

This symbol	*Identifies*
➢	Instructions for performing special functions.
Note	Additional information that may be of value.
Tip	Tips or suggestions about using a particular feature.
Caution ➤	Important information you need to know about a feature or procedure.

Related Publications

For more information, see the following publications:

H&W manuals

- *CPMS/SYSD Installation Manual*
- *CPMS/SYSD Reference Manual*

IBM manuals

- *Interactive System Productivity Facility Dialog Developer's Guide and Reference,* Version 4 Release 2 for MVS (GC34-4486)
- *Interactive System Productivity Facility Dialog Management Services* (GC34-4021)
- *Interactive System Productivity Facility ISPF Dialog Management Guide,* Version 3 for MVS (GC34-4213)
- *Interactive System Productivity Facility ISPF Dialog Management Guide and Reference,* Version 3 Release 5 for MVS (GC34-4266)

Chapter 1
Introduction

SYSD/JFT (Job and File Tailoring) is an option available for both SYSD and CPMS. JFT is based on IBM's ISPF Dialog Manager Panel Display capability. Wherever possible, the same syntax has been applied to JFT. If you are familiar with ISPF, you will see many similarities. JFT lets you:

- Build panel images.
- Prompt and validate user input.
- Process information from the user by applying it to a skeleton file.
- Submit JCL to the internal reader or save it to an output file.

This chapter describes:

- JFT's components
- Storing panels, skeletons, and messages
- JFT's variables
- Executing JFT panels

Components of SYSD/JFT

JFT applications are built using three different components: panels, skeletons, and messages. This section describes each component.

Panels

Panels drive all processing in JFT. Typically panels are either menus that lead to other JFT panels or input/output panels that let the user submit jobs (JCL) or update files. Each JFT user has a default panel that is executed when he or she selects Option 8, Job/File Tailoring, from CPMS/SYSD's main menu. The default panel is defined on Option 0.3, Job/File Tailoring Parameters. This means you can build custom JFT panels for different groups and have them go directly to the panel for their specific use. For example, you can have payroll users go directly to the payroll JFT panel and developers go directly to a system utilities JFT panel.

The panel defines and assigns all variables, defines the panel layout, accepts and edits input from the panel, and performs any output. When developing panels, you control the panel's display attributes by defining fields as input, output, or text and by defining the fields as highlighted or normal. The panel's)ATTR section defines these attributes. The panel's)BODY section defines the way the panel looks when a user accesses it. The panel's)INIT section defines and initializes variables. The panel's)PROC section does all the processing each time the user presses **Enter** or a **PFn** key. The panel's)PROC section also defines all editing for data entered and controls other processing like displaying another panel, writing to a file, or submitting a job to the system.

Skeletons

Skeleton libraries contain members that define the JCL for submitting a job or the record and file layout for updating a file. You use the FILESKEL and SUBSKEL commands in the)PROC section to access skeleton libraries. JFT processes these two commands by scanning the JFT skeleton file concatenation for the member name you specify on the command. JFT uses the first library that has a member name that matches. Your SYSD administrator defines the library concatenation.

JFT automatically passes variables from the panel to the skeleton member. JFT substitutes the data the user enters on the panel in the skeleton before it writes the skeleton to a file or the internal reader. For example, you may have a batch job that uses a date parameter to extract data from a master file. You can set up the JFT panel to ask the user to enter the date, pass that date to the skeleton member for substitution in the PARM parameter of the EXEC JCL statement, and submit the job.

Messages

Message libraries contain members that define the customized messages JFT uses to override the generic default messages provided. For example, if the user enters an invalid date, you can have JFT display a customized message that gives the user more specific information about what is wrong with the date. You define these messages and then issue them from the JFT panel.

You can also put JFT variables in your customized messages. JFT replaces the variable name with the variable's assigned value before displaying the message on the user's terminal.

Storing Panels, Skeletons, and Messages

Panels, skeletons, and messages are stored in partitioned datasets (PDSs). You must store each panel and skeleton in a separate PDS member. You can store several messages in a single PDS member.

Your SYSD administrator can concatenate panel, skeleton, and message PDSs and have JFT search them using a top down approach. JFT searches the first file specified in the concatenation, followed by the second file, and so on until it either reaches the end of the concatenation list or finds a matching member name. Your CPMS/SYSD administrator determines the order JFT searches the libraries.

Your SYSD administrator may have set up the system so users can specify the panel, skeleton, and message concatenations. This is done by defining the user profile dataset variables in the JFT dataset concatenations. If the variables are defined, the user can specify the dataset names JFT uses to search for panels, messages, and skeletons on Option 0.3, Job/File Tailoring Parameters. This lets the user or programmer dynamically change the dataset names.

JFT Variables

JFT lets you define your own variables in the panel's)BODY,)INIT, and)PROC sections. You can also use the system, user profile, and control variables described in Chapter 3, Variables. You can use any of these variables in the panels, skeletons, or messages. You can also pass JFT variables to CICS programs.

Executing JFT Panels

Once you have saved a panel in a PDS member, JFT can try to execute it. You do not have to compile it first. JFT reports any errors back to you when you execute the panel. You can edit the PDS member, correct the errors, and test the panel again.

To execute a JFT panel, the user must select Option 8, Job/File Tailoring, from the CPMS/SYSD main menu. Once in JFT, the user can execute JFT panels in one of three ways:

- Specify the panel as the default panel name on Option 0.3, Job/File Tailoring Parameters. This panel is automatically displayed when the user selects Option 8.
- Use the PANEL statement in one panel to call another panel. See the &ZSEL system variable on page 21 for more information about calling a panel.
- Type **EXEC panel_name** in the *Input* field in JFT and press **Enter**. To use this option, the *Execute Any Panel* field on Option 0.3, Job/File Tailoring Parameters, must be set to **Y** (Yes).

Tip

The EXEC command is very helpful because you do not have to update the default panel or test menus before you execute the new panels as you are developing them.

Chapter 2
Panel Sections

A SYSD/JFT panel can have up to five different sections. They are:

-)ATTR – Defines the attribute characters used in the)BODY section. These attribute characters define if fields are input, output, or text. They also define the intensity of the characters when JFT displays the panel: bright, normal, or dark.
-)BODY – Defines the panel's appearance.
-)INIT – Contains the logic statements JFT executes the first time a user accesses the panel.
-)PROC – Contains the logic statements JFT executes each time a user accesses the panel *except* the first time.
-)END – Marks the end of the panel.

Option 0.3, Job/File Tailoring Parameters, on the CPMS/SYSD main menu lets you specify which panel is displayed when the user first accesses the JFT option. The default is the SYSDO8 panel.

This chapter describes each panel section and its parameters.

)ATTR Section

The)ATTR section defines the characters that represent attribute bytes. You use these attribute characters in the)BODY section to define the panel's appearance. If you are only going to use the default attribute characters, you can omit the)ATTR statement altogether. The format of the attribute statement is:

```
char TYPE(TEXT|INPUT|OUTPUT) INTENS(LOW|HIGH|NON)
```

Note

You *cannot* continue statements in the)ATTR section.

This parameter	***Specifies***
char	The 1-byte character that represents the attribute byte. This is a special character that is *not* displayed on the panel when the user accesses it. However, the attribute character does take up a physical space on the panel when JFT displays it.
TYPE	The type of field this attribute byte generates.

Specify	***To define***
TEXT	A protected text field where JFT displays static information like menu options or field descriptions. This type of field cannot contain variables.
INPUT	An unprotected field where users can type data.
OUTPUT	A protected field where JFT displays the values of variables from within JFT.

INTENS	The intensity of the field when JFT displays the panel.

Specify	***To define***
LOW	A normal intensity field.
HIGH	A high intensity field.

Specify	*To define*
NON	A field that is not displayed on the panel. For example, you may want to define a field where the user types a password and not have the characters displayed as the user types them.

For example, if you specify:

```
)ATTR
 ¬ TYPE(INPUT) INTENS(LOW)
@ TYPE(OUTPUT) INTENS(HIGH)
```

A field defined in the)BODY section with a not sign (¬) is an input field the user can type data in. JFT displays the field at normal intensity. A field defined in the)BODY section with an "at sign" (@) is an output field JFT displays variables in. JFT displays the contents of the field as highlighted text.

Default Attribute Characters

The following are the predefined attribute character defaults:

```
% TYPE(TEXT) INTENS(HIGH)
+ TYPE(TEXT) INTENS(LOW)
_ TYPE(INPUT) INTENS(HIGH)
```

Overriding the Default Attribute Characters

To override the default attribute characters, specify the **DEFAULT(characters)** keyword, where **characters** are the new attribute characters, on the)ATTR statement.

To only override one default attribute character, you must specify all three attribute characters even though only one changes. For example, to change the default percent sign (%) to a pound sign (#), specify:

```
)ATTR DEFAULT(#+_)
```

If you do not need to change the default attribute characters, do not specify the DEFAULT keyword on the)ATTR statement.

)BODY Section

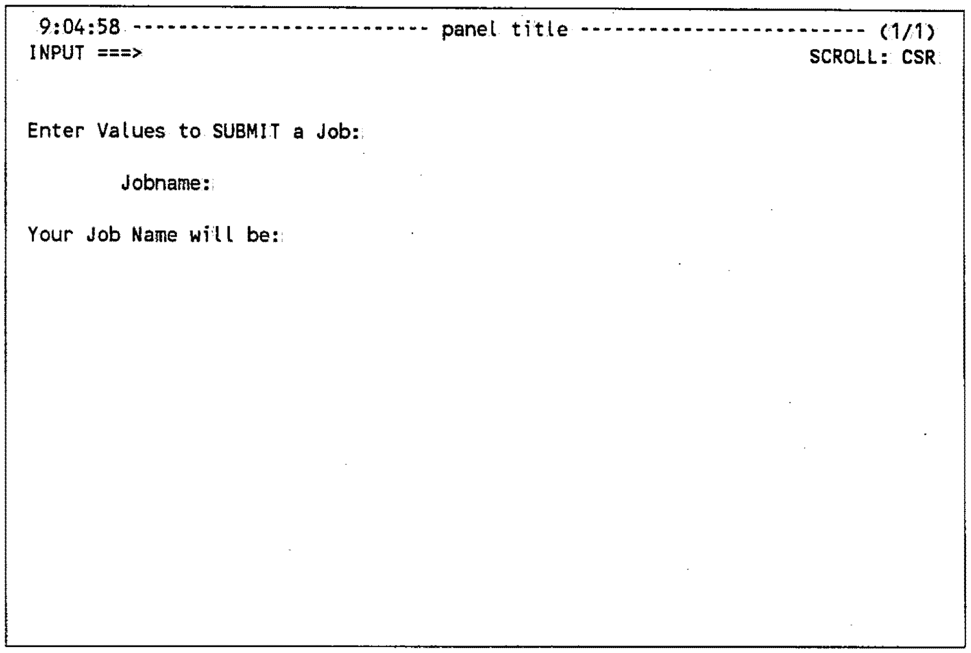
The)BODY section defines how a panel looks when JFT displays it. The following syntax rules apply to the)BODY section:

- The first three lines of the panel are fixed. The first line contains the panel title, time, and partition ID. The second line contains the *Input* field where the user enters commands and the *Scroll* field where the user defines the type of scrolling performed. The third line is where JFT displays the long system messages.
- The)BODY section defines lines 4 through 24 on the panel. A panel cannot have more than 21 lines. (MOD2 support only)
- If there is a blank line in the)BODY section, JFT displays a blank line on the panel. If you do not use all 21 lines, you do not have to add blank lines at the end of the)BODY section.
- Each line must begin with an attribute character, but the attribute character does not have to be in column 1.
- The attribute character defines the beginning of a field on the panel. The next attribute character or the end of the line defines the end of the field. The length of the field is the number of bytes between the attribute characters or between the beginning attribute character and the end of the line.
- If the field is defined as TYPE(TEXT), JFT displays all the characters in the field on the panel.
- If the field is defined as either TYPE(INPUT) or TYPE(OUTPUT), you *must* specify a variable name following the attribute character. A variable name that follows a TYPE(INPUT) or TYPE(OUTPUT) attribute *must not* include the ampersand (&); an ampersand is implied.
- Comments are not allowed in the)BODY section.

Example

```
)ATTR
 ¬ TYPE(OUTPUT) INTENS(HIGH)
)BODY

%Enter Values to SUBMIT a Job:%

        %Jobname:_JOBN    %

%Your Job Name will be:¬JOBN
)END
```

This example generates the following panel:



```
 9:04:58 ------------------------- panel title ------------------------ (1/1)
INPUT ===>                                                        SCROLL: CSR


Enter Values to SUBMIT a Job:

        Jobname:

Your Job Name will be:
```

This panel has:

- A protected text field beginning in column 2 of row 5 that displays *Enter Values to SUBMIT a Job:* in highlighted characters.
- A protected text field beginning in column 10 of row 7 that displays *Jobname:* in highlighted text.
- An 8-byte input field beginning in column 19 of row 7. The user types the job name in this field. JFT assigns the value the user types in this input field to the JOBN variable.
- A protected text field beginning in column 2 of row 9 that displays *Your Job Name will be:* in highlighted text.
- A 55-byte output field beginning in column 25 of row 9. There is no ending attribute character so the end of the output field is the end of the line. JFT displays the value of the JOBN variable in this field.

)INIT Section

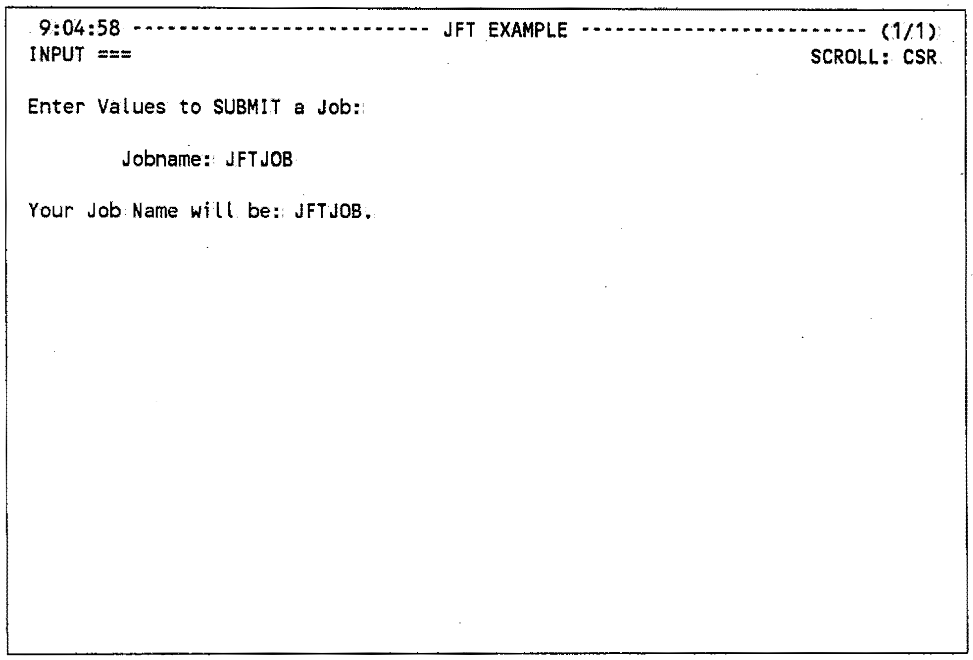
The)INIT section is optional. It defines the logic JFT *only* executes the first time a user accesses the panel. When returning to a previous panel, JFT executes the)INIT section for the previous panel because JFT considers it a first-time access. There are no parameters on the)INIT statement.

The most common use of the)INIT section is to declare and initialize variables. This is useful when you set default values for the user. You can also include logic statements in the)INIT section to conditionally set variable values. See Chapter 4, Logic Statements, for more information about the statements you can use in the)INIT section.

Example

```
)ATTR
 ¬ TYPE(OUTPUT) INTENS(HIGH)
)BODY

%Enter Values to SUBMIT a Job:%

        %Jobname:_JOBN    %

%Your Job Name will be:¬JOBN

)INIT
 .SYSDTTL = 'JFT EXAMPLE'
 &JOBN = JFTJOB

)END
```

This example generates the following panel:



```
 9:04:58 ------------------------- JFT EXAMPLE ------------------------ (1/1)
INPUT ===                                                            SCROLL: CSR

Enter Values to SUBMIT a Job:

        Jobname: JFTJOB

Your Job Name will be: JFTJOB.
```

Note what the)INIT section has done:

- The .SYSDTTL control variable defines the panel title as *JFT Example.* JFT centers the title on the first line of the panel.
- The *Jobname:* field now contains the default value assigned to the &JOBN variable in the)INIT section.
- JFT also displays the value of the &JOBN variable in the *Your Job Name will be:* field.

The user can change the &JOBN variable by typing a new job name in the *Jobname* field. When the user presses **Enter**, JFT changes the value displayed in the *Your Job Name will be:* field to the value the user typed.

)PROC Section

The)PROC section is optional. It defines the logic JFT executes every time a user accesses the panel *except* the first time. There are no parameters on the)PROC statement.

The)PROC section is where you do most of the programming in JFT. You can assign values to variables, test conditions, execute other panels, submit JCL to the operating system, or write records to files. An advanced feature of JFT lets you link to a CICS program and perform additional processing. This is done using the LINK statement.

See Chapter 4, Logic Statements, for more information about the statements you can use in the)PROC section.

Example

```
)ATTR
 ¬ TYPE(OUTPUT) INTENS(HIGH)
)BODY

%Enter Values to SUBMIT a Job:%

        %Jobname:_JOBN    %

%Your Job Name will be:¬VALJOBN

)INIT
 .SYSDTTL = 'JFT EXAMPLE'
 &JOBN = JFTJOB

)PROC
 &VALJOBN = '*ERROR*'      /*Assume Job Name invalid*/
 VER (&JOBN,NAME)          /*Check that Job Name is a valid member name*/
 IF (.MSG = &Z)            /*If no Error messages*/
     &VALJOBN = &JOBN      /*Move Job Name to a valid field*/
     SUBSKEL (&JOBN)
)END
```

Note what the)PROC section does:

- The)PROC section starts with an assignment statement that assigns the value ***ERROR*** to the &VALJOBN variable.
- The)PROC section shows how you can add comments that explain what each statement is doing. All text following the /* is a comment.
- The VER (&JOBN,NAME) statement checks to make sure the job name the user enters is a valid member name. The VER statement lets you easily validate common field formats, including:
    - non-blank (must be entered)

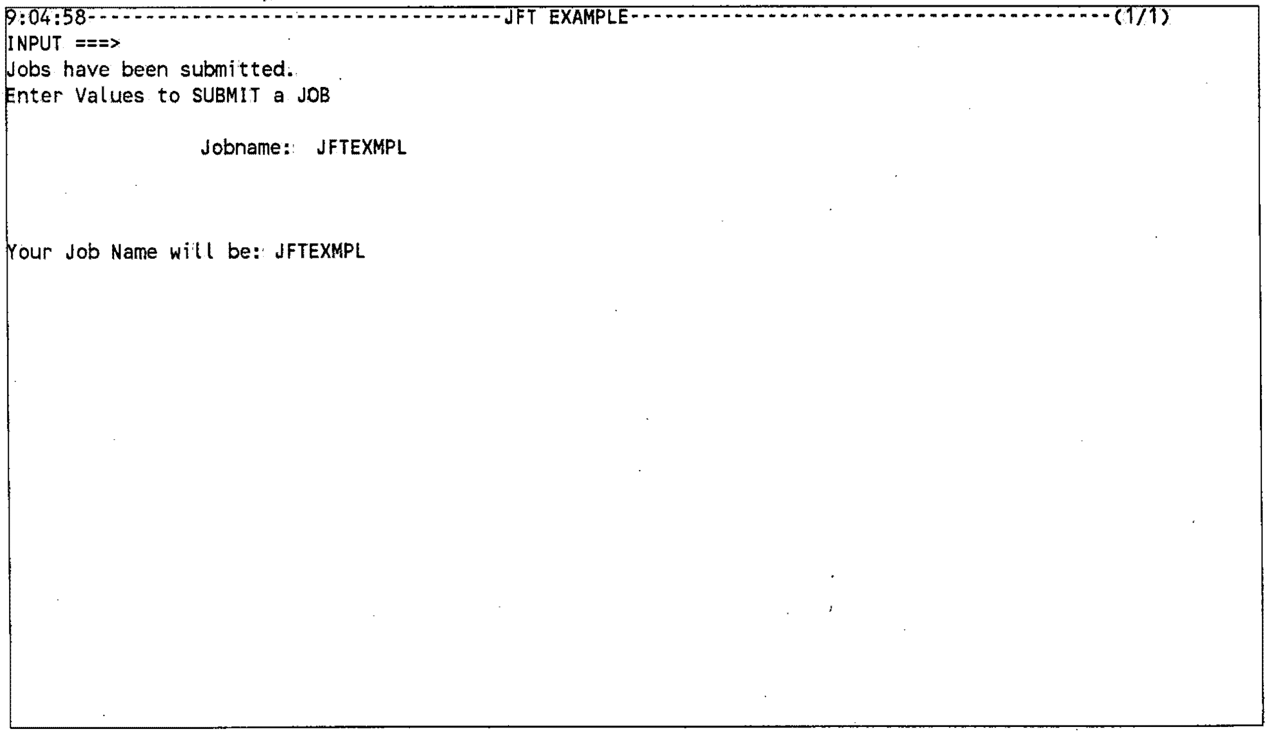
- alphabetic
- numeric
- hexadecimal
- PICTURE
- member name
- dataset name
- range
- list

♦ The next statement is an IF structure. JFT checks the .MSG control variable to see if an error was found. The &Z system variable tests the .MSG variable to see if it is empty or contains null values. So the IF statement says that if the .MSG control variable is empty, no errors were found.

If the IF statement is false, which means the member name is invalid, JFT executes the statement beginning in the column that is less than or equal to the IF statement column. In this case, JFT executes the)END statement.

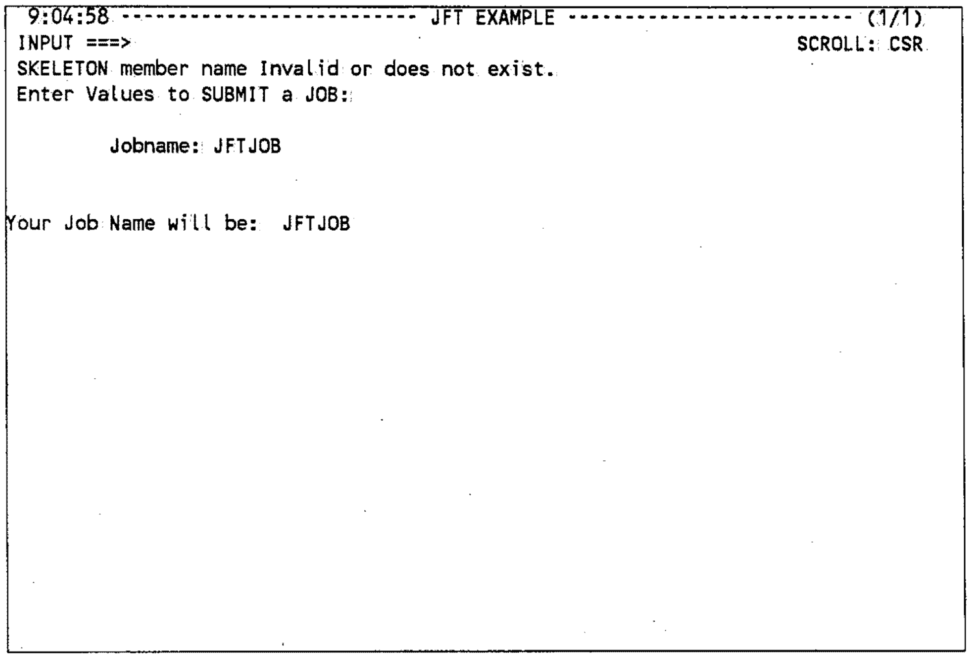
If the IF statement is true, which means the member name is valid, JFT executes the line right after the IF statement. In this case, JFT executes the assignment statement, which assigns the value of the &JOBN variable to the &VALJOBN variable.

♦ Assume the operator enters JFTEXMPL as the job name. If the entered name is valid, the SUBSKEL statement tells JFT to submit the skeleton JCL member identified by the JOBN variable. JFT does this by looking for a member in the JFT skeleton library concatenation called JFTEXMPL. If JFT finds a matching member, it substitutes any variables found and then submits the JCL to the operating system. The following is an example of what the panel looks like after a user enters a valid job name of JFTEXMPL:



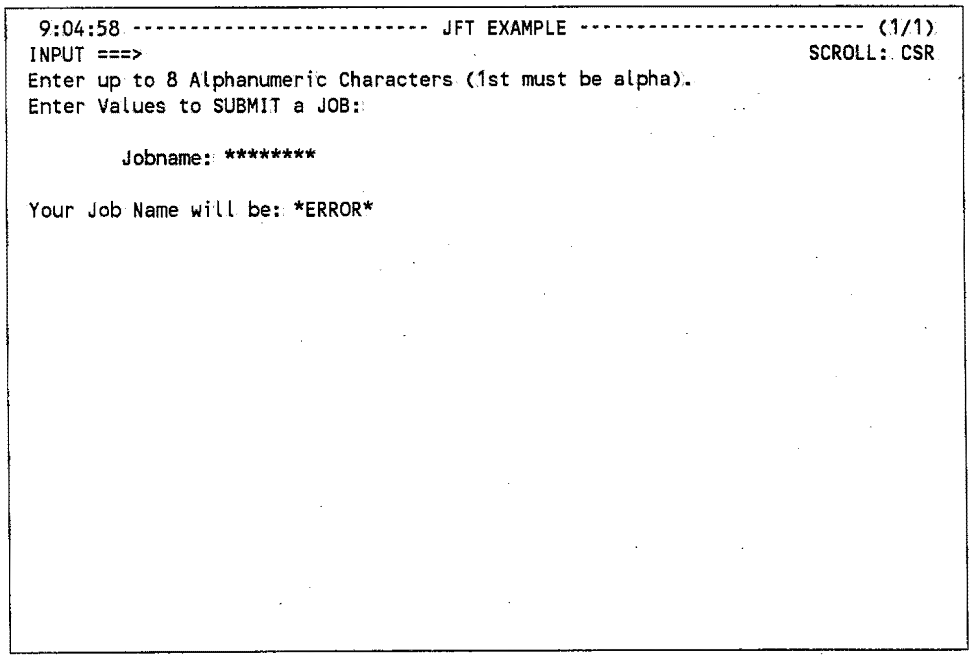
```
9:04:58-----------------------------------JFT EXAMPLE---------------------------------------(1/1)
INPUT ===>
Jobs have been submitted.
Enter Values to SUBMIT a JOB

                Jobname:  JFTEXMPL



Your Job Name will be: JFTEXMPL
```

If JFT does not find a matching member in any skeleton library, it displays the following error message:



```
 9:04:58 -------------------------- JFT EXAMPLE ------------------------- (1/1)
INPUT ===>                                                        SCROLL: CSR
SKELETON member name Invalid or does not exist.
Enter Values to SUBMIT a JOB:

        Jobname: JFTJOB


Your Job Name will be:  JFTJOB
```

If the user entered an invalid job name, like ********, JFT displays the following panel:



```
 9:04:58 -------------------------- JFT EXAMPLE ------------------------- (1/1)
INPUT ===>                                                        SCROLL: CSR
Enter up to 8 Alphanumeric Characters (1st must be alpha).
Enter Values to SUBMIT a JOB:

        Jobname: ********

Your Job Name will be: *ERROR*
```

)END Section

The)END section defines the end of the panel. JFT ignores anything following the)END statement. There are no parameters on the)END statement.

Chapter 3
Variables

This chapter describes the variables you can use in the)INIT,)PROC, and)BODY sections. There are four types of variables:

- System variables provide information from the system, like the date, time, and data entered in the *Input* field.
- Profile variables provide access to the information stored in the CPMS/SYSD user file. You can get information from the file, like the user ID or default printer, and use it in JFT processing.
- Control variables let you change the message and cursor position during processing. You typically use these variables to override the defaults for these functions. You can also use a control variable to define the panel title.
- User variables store values during JFT execution. You define these variables and can change them during processing.

System Variables

System variables provide information from the system, like the date, time, and data entered in the *Input* field. System variables begin with the letter Z. You can also update the &ZSEL variable to control JFT processing.

&Z

The &Z system variable contains a null or blank string. You can use the &Z variable to initialize another variable or use it in an IF statement to do a comparison.

&ZCMD

The &ZCMD system variable contains the characters the user typed in the *Input* field.

&ZCSR

The &ZCSR system variable contains the cursor position of the *Input* field in **rrccc** format, where **rr** is the row and **ccc** is the column.

&ZDATE

The &ZDATE system variable contains the 8-character date in **yy/mm/dd** format.

&ZDATESTD

The &ZDATESTD system variable contains the 8-digit date in **yyyymmdd** format.

&ZDAY

The &ZDAY system variable contains the 2-digit day of the month in **dd** format.

&ZJDATE

The &ZJDATE system variable contains the 6-character Julian date in **yy.ddd** format.

&ZJ4DATE

The &ZJ4DATE system variable contains the 8-character Julian date in **yyyy.ddd** format.

&ZMONTH

The &ZMONTH system variable contains the 2-digit month in **mm** format.

&ZSEL

You can use the assignment statement to set the &ZSEL system variable. If you set &ZSEL in the)PROC section, SYSD/JFT invokes a new panel when it is done processing the)PROC section. The syntax of the &ZSEL variable is:

```
&ZSEL = PANEL(panel_name)
```

For example:

```
&ZSEL = TRANS(TRUNC(&ZCMD,1),1,'PANEL(TESTP1)',2,'PANEL(TESTP2)')
```

This example shows how you can use logic statements on the right side of an assignment statement. If the user selects Option 1, JFT assigns **PANEL(TESTP1)** to &ZSEL. JFT sends the user to another panel from a menu panel that contains two options.

&ZSTDYEAR

The &ZSTDYEAR system variable contains the 4-digit current year in **yyyy** format.

&ZTIME

The &ZTIME system variable contains the 5-character time in **hh:mm** format.

&ZYEAR

The &ZYEAR system variable contains the 2-digit year in **yy** format.

Profile Variables

Profile variables let you access the information stored in the CPMS/SYSD user file. You can get information, like the user ID and default printer, and use it in JFT processing. Profile variables begin with a dollar sign ($). The following are some of the most commonly used profile variables. See Appendix A, CPMS/SYSD Menu System Variables, for a list of additional profile variables.

&$DEPART

The &$DEPART profile variable contains the user's 1- to 29-character department name.

&$DESTID

The &$DESTID profile variable contains the output destination ID from Option 0.1, JES/List Parameter Destinations. The length of this variable is 10. You can use the &$DESTID profile variable to substitute a destination ID in the JCL output parameter.

&$NAME

The &$NAME profile variable contains the user's 1- to 29-character name.

&$TITLE

The &$TITLE profile variable contains the user's 1- to 29-character title.

&$USER

The &$USER profile variable contains the user's 1- to 11-character user ID.

Control Variables

Control variables let you change the message and cursor position during processing. You typically use these variables to override the defaults for these functions. You can also use control variables to define the panel title and set the .TRAIL variable with the remainder of a TRUNC function. Control variables begin with a period (.).

.CURSOR

You can set the .CURSOR control variable to a **variable_name** without the ampersand (&). This positions the cursor in the field associated with **variable_name**. You can also use .CURSOR to set the cursor to the field on a VER statement that fails its test.

Note

If the cursor position has been previously set, .CURSOR does not set the cursor position. To reset the cursor position, you must use the .CURSORF control variable.

For example, if you specify:

```
.CURSOR = FNAME
```

JFT positions the cursor in the *FNAME* field when it displays the panel. The default position is the *Input* field.

.CURSORF

The .CURSORF control variable provides the same functions as the .CURSOR control variable except .CURSORF overrides any previous positioning of the cursor. Be careful when you use .CURSORF to set the cursor position so you do not accidentally override a cursor position.

```
.CURSORF = FNAME
```

.MSG

You can set the .MSG control variable to any message ID. JFT displays the long message associated with the message ID. You can also use .MSG to display a specific message ID when a VER statement fails its test.

Note

If the message ID has been previously set, .MSG does not reset the message ID. To reset the message ID, you must use the .MSGF control variable.

For example, if you specify:

```
.MSG = MSG001
```

JFT displays the text associated with MSG001.

.MSGF

The .MSGF control variable provides the same functions as the .MSG control variable except .MSGF overrides any previous setting of the message ID. Be careful when you use .MSGF to set the message ID so you do not accidentally override a message.

```
.MSGF = MSG001
```

.SYSDTTL

The .SYSDTTL control variable defines the panel title. You use the assignment statement to set the .SYSDTTL control variable. For example, if you specify:

```
.SYSDTTL = JFT Main Menu
```

JFT centers *JFT Main Menu* on the first line of the panel.

.TRAIL

JFT sets the value of the .TRAIL control variable as a result of a TRUNC statement. The .TRAIL variable contains the characters left over from the last TRUNC statement performed.

For example, if you specify:

```
&TEMP = ABCDEF
&TEMP1 = TRUNC(&TEMP,1)
```

JFT assigns the value **A** to the &TEMP1 variable and assigns the value **BCDEF** to the .TRAIL variable. See the TRUNC statement on page 40 for more information.

User Variables

User variables are variables you define. You can assign values to them and change the values during JFT execution. In the)INIT and)PROC sections, user variables must begin with an ampersand (&). In the)BODY section, user variables must not begin with an ampersand; the ampersand is implied. User variable names must be between 1 and 8 characters long.

Caution

Do not begin a user variable name with a dollar sign ($), the letter Z, or a period (.) because these characters identify system, profile, or control variables.

Chapter 4
Logic Statements

Logic statements control the processing that occurs when a user accesses a panel. You can specify logic statements in both the)INIT and)PROC sections. This chapter describes the syntax rules for all the logic statements and the syntax for each one.

Syntax Rules

Logic statements must follow these syntax rules:

- A /* means the rest of the line is a comment. You can specify an */ at the end of the comment, but it is not required.
- You can continue parameters on the next line if they follow the end of a parameter. For example:

```
VER (&TEST,NB,LIST,
       item1,
       item2)
```

- You can continue quoted strings by specifying a plus sign (+) as the last character of the line. The continuation line starts at the first non-blank character. For example, if you specify:

```
&TEST  = 'THIS IS LINE1 +
        AND THIS IS LINE2'
```

JFT assigns the value of **THIS IS LINE1 AND THIS IS LINE2** to the &TEST variable.

Assignment Statement

The assignment statement evaluates an expression and assigns the value of the expression to a variable. The syntax of the assignment statement is:

```
variable1 = variable2|literal|'literal'|function
```

This parameter	*Specifies*
variable1	A variable name preceded by an ampersand (&). The equal sign (=) *must* follow the variable name. JFT assigns the value of the expression on the right side of the equal sign to this variable.
variable2	A variable name preceded by an ampersand (&). JFT assigns the value of this variable to **variable1**.
literal	A literal you want to assign to **variable1**.
'literal'	A literal containing substitutable variables. JFT assigns this literal to **variable1**. See "Concatenation Rules" for more information about specifying variables in a literal.
function	A TRANS or TRUNC logic statement. See the TRANS statement on page 38 and the TRUNC statement on page 40 for more information about their parameters.

Concatenation Rules

The following concatenation rules apply to assignment statements:

- A double ampersand (&&) represents a single ampersand (&). A single ampersand (&) followed by a blank also represents a single ampersand (&).
- User variable names must begin with an ampersand (&). Variable names are delimited by any non-alphanumeric character.
- JFT ignores trailing blanks when doing a substitution.
- For an apostrophe or a quote within a quote, use two apostrophes ('') or quotes ("") to represent a single apostrophe (') or quote (").

- A single period (.) following a variable name on the right side of an equal sign concatenates the character string following the variable name with the variable. Two periods (..) following a variable name represent a single period (.). For example, if you specify:

```
&CITY = BOISE
&ADDR = '&CITY. ID'
```

JFT assigns **BOISE ID** to the &ADDR variable.

Note

If you specify a period as part of a literal string, you only have to specify one period to represent a single period.

Examples

The following are several examples of the assignment statement.

Example 1

```
&FNAME = JODY
```

JFT assigns the value **JODY** to the &FNAME variable.

Example 2

```
&LNAME = LANTZ
```

JFT assigns the value **LANTZ** to the &LNAME variable.

Example 3

```
&LNAME2 = &LNAME
```

JFT assigns the value of the &LNAME variable to the &LNAME2 variable. If JFT executed the assignment statement from Example 2 before this statement, JFT assigns the value **LANTZ** to the &LNAME2 variable.

Example 4

```
&MNAME = 'M'
```

JFT assigns the value **M** to the &MNAME variable. The apostrophes are optional.

Example 5

```
&LNAME = LANTZ
&FNAME = JODY
&MNAME = M
&FULLNAME = '&LNAME, &FNAME &MNAME..'
```

JFT assigns the value **LANTZ, JODY M.** to the &FULLNAME variable.

Note

The two periods at the end of the last assignment statement result in only one period after the middle initial. If a period follows a variable name, JFT requires two periods to represent one period.

Example 6

```
&LNAME = LANTZ
&LNAMEF3B = TRUNC(&LNAME,3)
```

JFT assigns the value **LAN** to the &LNAMEF3B variable. JFT assigns the rest of the characters, which are **TZ**, to the .TRAIL control variable.

FILESKEL Statement

The FILESKEL statement saves a member in a dataset using a skeleton member as a model. The syntax of the FILESKEL statement is:

```
FILESKEL(DSN(variable_name|literal),NAME(variable_name|literal),
        SKEL(variable_name|literal),VOL(variable_name|literal),
        REPL|NOREPL|MOD)
```

This parameter	Specifies
DSN	The name of the dataset you want to save the member in. You can specify either a literal or a variable name preceded by an ampersand (&).
NAME	The member name if the value of the DSN parameter is a PDS. You can specify either a literal or a variable name preceded by an ampersand (&). For a sequential dataset, the member name is optional.
SKEL	The member name of the skeleton file you want to use as a model. You can specify either a literal or a variable name preceded by an ampersand (&).
VOL	The volume serial number where the dataset resides. This parameter is optional. You can specify either a literal or a variable name preceded by an ampersand (&).
REPL	Replace the member in the dataset if it already exists. This is the default.
NOREPL	Terminate the save if the member already exists in the dataset.
MOD	Add the data specified by the FILESKEL parameters to the end of the dataset JFT writes. This parameter only supports sequential datasets.

Example 1

```
FILESKEL(DSN(&HOLDDSN),NAME(MEMBER1),SKEL(&SKELMEM))
```

JFT saves the member called MEMBER1 in the dataset specified by the &HOLDDSN variable using the skeleton member specified by the &SKELMEM variable.

Example 2

```
FILESKEL(DSN(XYZ.ABC),NAME(MEMBER2),SKEL(MEM10),NOREPL)
```

If MEMBER2 does not already exist, JFT saves the member called MEMBER2 in the dataset called XYZ.ABC using the skeleton member called MEM10.

Example 3

```
FILESKEL(DSN(JFT.TXN.FILE),SKEL(EMPREC),MOD)
```

The skeleton member EMPREC defines the record layout. JFT adds records to the end of the file called JFT.TXN.FILE using the layout defined in EMPREC.

IF Statement

The IF statement evaluates an expression and, depending on if the expression is true or false, executes a statement or group of statements. The syntax of the IF statement is:

```
IF(operand1 operator operand2)
```

The parameter	*Specifies*
operand1	A user variable name preceded by an ampersand (&) or a control, system, or profile variable.
operator	The operator JFT uses to compare the values of **operand1** and **operand2**. Valid operators are: **=** **¬=** **GT** **GE** **LT** **LE**
operand2	A user, control, system, or profile variable name or a list of literals.

When used with IF statements, logic statements are sensitive to their column position. If the condition of an IF statement is true, JFT executes the next statement after the IF statement. If the IF statement is false, JFT executes the next statement that has a column position less than or equal to the column position of the IF statement.

Example 1

```
IF(&COND1 = &COND2)
  STM1
 STM2
STM3
```

If the condition in the IF(&COND1 = &COND2) statement is true, JFT executes STM1, STM2, and STM3. If the condition is false, JFT only executes STM3 because it has a column position (column 1) that is less than or equal to the IF(&COND1 = &COND2) statement (column 1).

Example 2

```
IF(&COND1 = &COND2)
  IF(&COND3 = YES,NO)
   STM1
  STM2
 STM3
STM4
```

If the condition in the IF(&COND1 = &COND2) statement is true, JFT executes the next IF statement. If the condition is false, JFT only executes STM4 because it has a column position (column 1) that is less than or equal to the column position of the IF(&COND1 = &COND2) statement (column 1).

If the IF(&COND3 = YES,NO) statement is true, which also means the IF(&COND1 = &COND2) statement was true, JFT executes STM1, STM2, STM3, and STM4. If the condition is false, JFT only executes STM2, STM3, and STM4 because STM2 has a column position (column 3) that is less than or equal to the column position of the IF(COND3 = YES,NO) statement (column 3).

LINK Statement

The LINK statement links from the JFT panel to a CICS program. The syntax of the LINK statement is:

```
LINK(PGM(variable_name|literal))
```

This parameter	*Specifies*
variable_name	A variable name preceded by an ampersand (&). The value of the variable must be the name of the CICS program you want JFT to link to.
literal	The name of the CICS program you want JFT to link to.

Note: A COMMAREA is not passed. If you need to communicate between the JFT panel and the CICS program, use the $CMVGET and $CMVPUT macros. See "Communicating Between Panels and CICS Programs" on page 54 for more information.

Example 1

```
LINK(PGM(&PGMLINK))
```

JFT links to the CICS program name specified as the value of the &PGMLINK variable.

Example 2

```
LINK(PGM(CICSPGM))
```

JFT links to the CICS program called **CICSPGM**.

SUBSKEL Statement

The SUBSKEL statement submits a skeleton member to the internal reader. The syntax of the SUBSKEL statement is:

```
SUBSKEL(variable_name|literal)
```

This parameter	*Specifies*
variable_name	A variable name preceded by an ampersand (&). The value of the variable must be the name of the skeleton member you want JFT to submit to the internal reader.
literal	The name of the skeleton member you want JFT to submit to the internal reader.

Note: JFT uses the SYSD DCT entry to submit the JCL to the internal reader.

Example 1

```
SUBSKEL(&SKELNM)
```

JFT submits the skeleton member specified as the value of the &SKELNM variable.

Example 2

```
SUBSKEL(MEMBER1)
```

JFT submits the skeleton member called **MEMBER1**.

TRANS Statement

The TRANS statement compares a variable to a value and assigns a corresponding value to another variable. The TRANS statement is used in an assignment statement. The syntax of the TRANS statement is:

```
result=TRANS(variable_name,value,new_value,...,value,new_value)
```

This parameter	*Specifies*
variable_name	A variable name preceded by an ampersand (&). This parameter must be specified first.
value	The value you want to compare **variable_name** to.
new_value	The value you want to assign to the **result** variable.
result	The variable the new value will be assigned to.

The parameters that follow **variable_name** *must* be paired values. If you specify **value**, you must also specify **new_value**.

JFT compares **variable_name** to each **value**. If the two values are equal, JFT assigns **new_value** to **result**.

An "anything else" condition occurs if **variable_name** does not match any of the **value** values. You can define an "anything else" condition by specifying an asterisk ('*') for **value** in the last set of paired values. If you also specify '*' for **new_value**, JFT stores the variable being assigned untranslated. If you do not specify '*' for **new_value**, JFT stores the value as specified.

Example 1

```
&ABC = TRANS(&DEF,1,ONE,2,TWO,3,THREE,' ',' ','*',NONE)
```

If &DEF contains **2**, &ABC contains **TWO**. If &DEF contains a blank, &ABC contains a blank. If &DEF contains **5**, &ABC contains **NONE**.

Example 2

```
&ABC = TRANS(&DEF,1,ONE,2,TWO,3,THREE,' ',' ','*','*')
```

Everything in Example 1 is also true for this example, except if &DEF contains **5**, &ABC contains **5**.

TRUNC Statement

The TRUNC statement truncates a variable at a specific position or character. The syntax of the TRUNC statement is:

```
TRUNC(variable_name column|character)
```

This parameter	*Specifies*
variable_name	A variable name preceded by an ampersand (&). This parameter must be specified first.
column	The column position where you want to truncate the variable.
character	The character where you want the variable truncated. The truncation occurs where this character appears first in the variable.

After JFT truncates **variable_name**, it assigns any remaining characters to the .TRAIL control variable. If you truncate **variable_name** based on column position, JFT assigns the rest of the characters to .TRAIL. If you truncate **variable_name** based on a character, JFT assigns all the characters *after* the truncation character to .TRAIL and drops the truncate character.

Example 1

```
&ABC = TRUNC(&DEF,8)
```

Assume &DEF contains **ABCDE:GHIJ**. After JFT executes the assignment statement, &ABC contains **ABCDE:GH**, &DEF does not change, and .TRAIL contains **IJ**.

Example 2

```
&ABC = TRUNC(&DEF,':')
```

Assume &DEF contains **ABCDE:GHIJ**. After JFT executes the assignment statement, &ABC contains **ABCDE**, &DEF does not change, and .TRAIL contains **GHIJ**. JFT drops the character you specified as the truncate character, which in this example is a colon (:).

Example 3

The TRUNC command can be used to parse a string. For example:

```
&FULLNAME = JODY M. LANTZ
&FNAME    = TRUNC(&FULLNAME,' ')
&TEMP     = .TRAIL
&INITIAL  = TRUNC(&TEMP,' ')
&LNAME    = .TRAIL
```

The first assignment statement assigns **JODY M. LANTZ** to the &FULLNAME variable. The second statement assigns **JODY** to the &FNAME variable and assigns **M. LANTZ** to the .TRAIL control variable. The third statement assigns the value of the .TRAIL control variable, which is now **M. LANTZ**, to the &TEMP variable. The fourth statement assigns **M.** to the &INITIAL variable. The last statement assigns **LANTZ** to the &LNAME variable.

VER Statement

The VER statement verifies the value of a variable. The syntax of the VER statement is:

```
VER(variable_name,NB,function,parameters,MSG=message_id)
```

This parameter	*Specifies*
variable_name	The variable name you want to verify.
NB	The value of the variable name cannot be blank. If specified, **NB** must follow **variable_name**. You can use this parameter with all the functions of the VER statement.
function	The type of information **variable_name** can contain.
parameters	One or more values JFT uses to verify **variable_name**.
message_id	The ID of the message you want to display if the VER statement fails its test. This parameter must be the last item on the list.

The following sections describe the syntax for the functions of the VER statement.

Caution

The NB parameter has the highest priority in a VER statement. If the NB parameter is not specified, the variable can have a blank or empty value regardless of the type of verification being done.

VER ALPHA

```
VER(variable_name ALPHA)
```

The **variable_name** must be alphabetic.

VER BIT

```
VER(variable_name BIT)
```

The **variable_name** must only contain **0**s or **1**s.

VER DSNAME

```
VER(variable_name DSNAME)
```

The **variable_name** must be a valid dataset name. All dataset qualifiers must begin with an alphabetic character and must be between 1 and 8 characters long.

VER HEX

```
VER(variable_name HEX)
```

The **variable_name** must contain hexadecimal numbers (0-9, A-F).

VER LIST

```
VER(variable_name LIST value1, value2, value3 ....)
```

The **variable_name** must be equal to one of the listed values.

VER NAME

```
VER(variable_name NAME)
```

The **variable_name** must be a valid member name. You can specify numeric characters, but the first character must be alphabetic. The length of the name must be at least 1 character and not more than 8 characters.

VER NB

```
VER(variable_name NB)
```

The **variable_name** must not be blank.

Note

You can use **NB** with all the functions of the VER statement. If **NB** is specified, it must follow the **variable_name**.

VER NUM

```
VER(variable_name NUM)
```

The **variable_name** must be numeric (0-9).

VER PICT

```
VER(variable_name PICT 'picture string')
```

The **variable_name** must match the corresponding type of character in **'picture string'**. Valid **'picture string'** characters are:

This character	*Specifies*
C	Any character.
A	Any alphabetic character (A-Z, #, $, @).
N	Any numeric character (0-9).
9	Any numeric character (0-9).
X	Any hex character (0-9, A-F).

Any other character represents itself.

VER RANGE

```
VER(variable_name RANGE n n)
```

The **variable_name** must be within the range of the two numbers specified.

Chapter 5
Skeleton and Message Files

JFT panels can access skeleton and message files to perform additional tasks. Skeleton files contain models or skeleton JCL members JFT can submit. Skeleton files can also contain model or skeleton record layouts JFT can use to update sequential files or PDS members. Message files contain messages of up to 80 characters JFT can display on the panel.

Skeleton Files

Skeleton files are PDS members or sequential files that contain skeleton JCL, control statements, or other data that require changing before they are saved to disk or submitted.

JFT retrieves the skeleton based on the dataset or member name specified in a FILESKEL or SUBSKEL statement. JFT scans records within the skeleton for variable names that are indicated by an ampersand (&). Any variables names that are found are replaced by the contents of the variable. Updated records that are blank after the subsitution are not written. The updated file is either saved to disk (FILESKEL) or submitted (SUBSKEL).

You can add control statements in a skeleton to help control the file-tailoring process. A control statement consists of a control character followed by a keyword. The control character must be in column 1 followed immediately by the keyword. If JFT does not recognize the keyword, it considers the line to be a comment. If the control character is followed by a blank, JFT considers the line to be a control statement.

Control Statements

)BLANK

The)BLANK statement inserts a specific number of blank lines in the output skeleton file. The syntax of the BLANK statement is:

```
)BLANK nn|&variable_name
```

This parameter	*Specifies*
nn	The number of blank lines you want to insert. The default is **1**. If you specify a number greater than 99, the default is **99**.
&variable_name	A variable name. The value of the variable must be the number of blank lines you want to insert.

)CM

The)CM statement is treated as a comment and ignored. The syntax of the CM statement is:

```
)CM
```

)DEFAULT

The)DEFAULT statement changes the default control character, which is a right parenthesis, to a different character. For example, if you specify:

```
)DEFAULT #
```

JFT changes the default control character to a pound sign (#). All control statements that follow the)DEFAULT statement must start in column 1 and begin with a pound sign (#)

)IM

The)IM statement copies in the contents of a member from the skeleton library concatenation. If you specify a variable name, the contents of the variable must be a member name. The syntax of the)IM statement is:

```
)IM member_name|&variable_name OPT NT
```

This parameter	*Specifies*
member_name	The name of the member you want JFT to copy.
&variable_name	The name of a variable. The value of the variable must be the name of the member you want JFT to copy.
OPT	Ignore the error and continue processing if JFT does not find the member. This parameter is optional.
NT	Copy the member. This parameter is optional. If this parameter is specified, JFT does not perform any file tailoring.

)TB

The)TB statement sets tabs for the output in a skeleton file. This statement is useful for positioning data in a certain column of a record. The syntax of the)TB statement is:

```
)TB column[A] ... column[A]
```

This parameter	***Specifies***
column	The position in the record where you want to set the tab.
A	The tab is absolute. This option must immediately follow the column number.

Example 1

Assume the following)TB statement is in the skeleton file:

```
)TB 16A 57
&FNAME! &LNAME! &MI
```

The exclamation point (!) is the tab character. It tells JFT to tab before inserting the field. As JFT moves data into the output area during file tailoring, it keeps track of where the data is to be positioned.

Assume the following input values:

```
&FNAME = DON
&LNAME = DAHL
&MI    = J
```

If you had not specified tabs in the skeleton file, the result would be:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--

DONDAHLJ
```

Since you did specify tabs in the skeleton file, the result would be:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--

DON            DAHL                                    J
```

The word *Don* starts in column 1, the word *Dahl* starts in column 16, and the initial *J* starts in column 57.

Example 2

Each time JFT finds an exclamation point (!), it advances to the next tab marker from the current output marker. If the current output marker is pointing at a tab, JFT jumps to the next tab unless the absolute (A) option is placed on the tab the output marker is pointing to.

Assume the following)TB statement is in the skeleton file:

```
)TB 10A 57A
&FNAME! &LNAME! &MI
```

Assume the following input values:

```
&FNAME = ELIZABETH
&LNAME = HARTE
&MI    = A
```

JFT puts the value of the &FNAME variable beginning in column 1.

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--

ELIZABETH
```

The output marker is now at position 10. The next instruction to JFT is to tab. Since the example has an absolute tab in column 10, JFT puts the value of the &LNAME variable beginning in column 10. The next instruction to JFT is to tab. JFT puts the value of the &MI variable in column 57. The result would be:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--

ELIZABETHHARTE                                          A
```

If you had not specified the A option for the column 10 tab in the skeleton file, the result would be:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--

ELIZABETH                                               HARTEA
```

Since no additional tabs were defined in the skeleton, JFT puts the value of the &MI variable immediately after the value of the &LNAME variable.

Message Files

Message files contain message members. Message members contain messages of up to 80 characters. You can have JFT display these messages on the panel.

The member name consists of the message ID through the second digit. For example, if you specify a message ID of **XYZ001A**, JFT searches for a message member name of **XYZ00**.

A message ID can be 4 to 8 characters long. The first 1 to 5 characters must be a valid member name. The rest of the characters, up to the maximum of 8, can be alphanumeric. For example, **AB00**, **AB001A**, and **ABCDE002** are valid message IDs.

Every message member contains at least two lines. The first line is the message ID beginning in column 1. The second line is the long message. You can enclose the message with apostrophes, but the apostrophes are not required. The message can contain variables. JFT replaces any variables in the message with their values. The following are examples of messages in the PDS member called XYZ00:

```
XYZ001A
'User must specify the correct value.'
XYZ002
The number '&NUM' is not valid.
```

To have JFT display these messages on the panel, use the following panel statements:

```
.MSG = message_id
.MSGF = message_id
VER (&FIELD,NB,MSG=message_id)
```

See the .MSG control variable on page 23, the .MSGF control variable on page 24, and the VER statement on page 42 for more information.

Chapter 6
Calling CICS Programs

This chapter explains how to call CICS programs from JFT panels. It also discusses how to access and update JFT variables from a CICS program.

Communicating Between Panels and CICS Programs

After the JFT LINK statement performs an EXEC CICS LINK command, the variables the JFT panel uses may need to be read and updated. JFT variables are managed by the CPMS/SYSD conversational manager and can only be updated using the $CMVGET and $CMVPUT macros. The $CMVGET macro reads the JFT variables into storage and the $CMVPUT macro updates the JFT variables. You must use the $CMVGET and $CMVPUT macros with Assembler in a SYSD-type program.

$CMVGET

The $CMVGET macro reads the JFT variables the program needs into storage. The syntax of the $CMVGET macro is:

```
$CMVGET (variable_name,length,location)
```

This parameter	*Specifies*
variable_name	The 8-byte variable name you want the $CMVGET macro to read. The variable name cannot include an ampersand (&). If the variable name is less than 8 bytes, it must be padded with blanks. You can specify either a data name that points to a variable name or the variable name itself enclosed with apostrophes.
length	The length of the variable. You can specify either a literal number or a number contained in a register.
location	Where you want the $CMVGET macro to put the contents of the variable. You can specify either the data name of a location in memory or an actual address specified in a register.

Examples

```
$CMVGET ('NAME    ',21,SAVENAME)
$CMVGET (SAMPKYWD,(R2),(R4))
```

$CMVPUT

The $CMVPUT macro updates JFT variables. The syntax of the $CMVPUT macro is:

```
$CMVPUT (variable_name,length,location,'F')
```

This parameter	*Specifies*
variable_name	The 8-byte variable name you want the $CMVPUT macro to update. The variable name must not include an ampersand (&). If the variable name is less than 8 bytes, it must be padded with blanks. You can specify either a data name that points to a variable name or the variable name itself enclosed with apostrophes.
length	The length of the variable. You can specify either a number or a number contained in a register.
location	Where you want the $CMVPUT macro to get the variable from. You can specify either the data name of a location in memory or an actual address specified in a register.
'F'	The conversational manager defines **variable_name** as a functional variable. All JFT variables are functional variables. This parameter is required.

Examples

```
$CMVPUT ('NAME    ',21,SAVENAME,'F')
$CMVPUT (SAMPKYWD,(R2),(R4),'F')
```

Calling CICS Programs from SYSD/JFT

This section provides an example of a JFT application called JFTADD that gets two numbers from a JFT panel. JFTADD shows how to get JFT variables in your CICS program, call subroutines in your CICS program, and return values to JFT from your CICS program. The panel validates the input to make sure it is numeric. The JFT panel validates the numbers and calls a CICS program to add them. The CICS program then returns the sum back to JFT. JFT displays the sum on the panel for the user.

Note

Appendix C, JFTADD Program, provides the complete code for JFTADD. The source is also in the JFT sample library. When you install SYSD/JFT, this program is copied to your SYSD.SOURCE library and is called JFTADD.

You can copy JFTADD and change it to create your own CICS programs to perform other functions. For example, you could have your CICS program read a VSAM file and return values back to JFT for display. You can also pass the JFT variables back to your CICS program and then update the VSAM file.

Calling the JFTADD Program

The following is the code for a panel that accepts and edits the input and then displays the result:

```
)ATTR
 ¬ TYPE(OUTPUT) INTENS(HIGH)
)BODY

%ENTER Number 1         _NUM1     %

%ENTER Number 2         _NUM2     %

%              ==================

%TOTAL        :      ¬SUM        %
)INIT
 .SYSDTTL = 'JFT Addition Link Program'
)PROC
 VER (&NUM1,NB,NUM)
 VER (&NUM2,NB,NUM)
 IF (.MSG = &Z)
   LINK(PGM(JFTADD))
)END
```

JFT variables are stored in character (EBCDIC) format. As a result, JFTADD must convert the data entered into the required format for processing. In this example, JFTADD is adding two numbers so the input must be converted to numeric values. To accomplish this, the VER statements verify that the fields are not blank (NB option) and that the characters entered in them are numeric (NUM option).

Then JFT checks the .MSG system variable to see if an error was found. JFT only calls JFTADD when NUM1 and NUM2 are both numeric, which means an error was not found.

The LINK command tells JFT to call JFTADD.

Note

The program JFT calls must be a CICS program and have a PPT table entry defined. If you change the program being called, you must issue a Newcopy command to CICS.

SYSD-like Programs

SYSD-like programs use the $PROC macro to mark the beginning of the program and subroutines. JFT calls these subroutines using assembly language call standards; that is, JFT passes the parameters using Register 1. The $PROC macro takes care of saving register information from the calling program. The $END macro marks the end of a program or subroutine and restores the registers for the calling program.

You can use the $RETURN macro in subroutines to set a return code. The format of the $RETURN macro is:

```
$RETURN RC=0
```

The calling program or routine can then check Register 15 to find out what the return code is. The panel cannot check the return code of the program it linked to. To accomplish this, you must define a JFT variable in your panel and use the $CMVPUT macro to set that variable in your CICS program.

Storing Data in JFT-like Programs

You can define user storage areas in one of three areas depending on how you will use the data.

Constant Data

You can define constant data, like tables, after the $END macro. For example:

```
         $END  RC=0
PI       DC    PL3'3.14'
```

Local Variables

Local variables are only addressable within the routine being processed. They are defined and addressable between the $PROC and $END macros. You can use the $DCL macro to define local variables.

The $DCL macro lets you allocate data areas in two ways. First, you can define data areas with individual $DCL macros for each variable. For example:

```
VALNUM $PROC
       $DCL NUMBEGN,FUL        POINTER TO PASSED VALUE
       $DCL NUMLEN,FUL         LENGTH OF THE FIELD PASSED
       $DCL PADLEN,FUL         LENGTH OF THE FIELD PASSED
       $DCL WORKNUM,CHAR(10)   WORK FIELD
```

Second, you can define them in a group with the $DCL ON and $DCL OFF macro parameters. For example:

```
&PGMNAME $PROC OPTIONS=(MAIN,COMMAND,GEN)
       $DCL ON
WPFLD1 DS  PL5       PACKED WORK AREA FOR NUMBER 1
WPFLD2 DS  PL5       PACKED WORK AREA FOR NUMBER 2
WPSUM  DS  PL6       PACKED WORK AREA FOR NUMBER SUM
       $DCL OFF
```

Global Variables

Global variables can be used anywhere in the program. You can define them in the program's Dynamic Storage Area. The following example shows three variables used in the JFTADD example:

```
                    ATITLE '** COMMAND LEVEL DYNAMIC STORAGE
**********************************************************************
*                            COMMAND LEVEL DYNAMIC STORAGE          *
**********************************************************************
*      AREA HEADER INFORMATION.
DFHEISTG  DSECT
          DFHEISTG ,
COMMBEGN EQU   *
*** PLACE DYNAMIC STORAGE AREAS HERE.
EBACNUM1 DS    CL9             FIRST NUMBER TO ADD
NUM1LEN  EQU   L'EBACNUM1      LENGTH OF NUMBER 1
EBACNUM2 DS    CL9             SECOND NUMBER TO ADD
NUM2LEN  EQU   L'EBACNUM2      LENGTH OF NUMBER 2
EBACSUM  DS    CL11            RETURN THE NUMBER BACK.
```

JFTADD Program

When JFT calls JFTADD, the first thing JFTADD does is get the JFT variables it will process. This is done using the $CMVGET macro. For example:

```
*
*       GET THE NUMBERS TO ADD
*
    $CMVGET ('NUM1     ',9,EBACNUM1)    FIRST NUMBER
    $CMVGET ('NUM2     ',9,EBACNUM2)    SECOND NUMBER
```

The $CMVGET macro copies the data from the NUM1 and NUM2 JFT variables into JFTADD's memory. The EBACNUM1 and EBACNUM2 variables are global variables in the program since they are stored in the Dynamic Storage Area, which means any routine in JFTADD can access them.

JFTADD then converts the data in EBACNUM1 and EBACNUM2 from character (EBCDIC) format to packed-numeric format, left justifying the data in the field. To convert the number, JFTADD calls the VALNUM routine to right justify and zero fill the field. The $CALL macro builds a parameter list and points Register 1 to that list before calling the VALNUM routine. After the VALNUM routine completes successfully, JFTADD adds the two numbers and converts EBACNUM1, EBACNUM2 , and EBACSUM back to character (EBCDIC) format.

To display the result back to the user, JFTADD uses the $CMVPUT macro to update the JFT variables. For example:

```
$CMVPUT ('NUM1     ',9,EBACNUM1,'F')  PASS NUMBER1 TO JFT PANEL
$CMVPUT ('NUM2     ',9,EBACNUM2,'F')  PASS NUMBER2 TO JFT PANEL
$CMVPUT ('SUM      ',11,EBACSUM,'F')  PASS SUM TO JFT PANEL
```

Note

You may want to try commenting out the $CMVPUT lines for the NUM1 and NUM2 variables. These two lines cause JFT to display the zero-filled, right-justified values to the user. If you comment them out, JFT displays the NUM1 and NUM2 variables as entered.

The $END macro indicates the end of the program and returns control to JFT. In this example, JFT ends processing and displays the results to the user. You can also continue with JFT processing logic based on the results returned from the CICS program.

Appendix A
CPMS/SYSD Menu System Variables

This appendix provides a list of the variables JFT can access from the CPMS/SYSD menu system.

Option 0.0, General Parameter Definitions

Variable name	*Length*	*Description*
$ADDR1	29	Address line 1
$ADDR2	29	Address line 2
$ADDR3	29	Address line 3
$ADDR4	29	Address line 4
$DEPART	29	User's department name
$NAME	29	User's name
$PSWD	9	User's password
$TITLE	29	User's title
$USER	11	User ID

Option 0.1, JES/List Parameter Definitions

Variable name	*Length*	*Description*
$CLASS	1	Job class selected for display
$DESTID	10	Destination
$DSPSTCS	1	Display TSO/STC
$JOBCRD1	72	JOB card 1

Variable name	*Length*	*Description*
$JOBCRD2	72	JOB card 2
$JOBCRD3	72	JOB card 3
$JOBCRD4	72	JOB card 4
$PREFIX	8	Job prefix selected for display
$PRINTER	4	Printer ID
$PRTPRTD	1	Printer display preference (CPMS or JES2)
$SYSINOK	1	View SYSIN JES datasets

Option 0.2, Program Function Key Definition

Variable name	*Length*	*Description*
$CLEAR	8	Clear key value
$PA1	8	PA1 key value
$PA2	8	PA2 key value
$PA3	8	PA3 key value
$PF01	8	PF1 key value
$PF02	8	PF2 key value
$PF03	8	PF3 key value
$PF04	8	PF4 key value
$PF05	8	PF5 key value
$PF06	8	PF6 key value
$PF07	8	PF7 key value
$PF08	8	PF8 key value
$PF09	8	PF9 key value
$PF10	8	PF10 key value
$PF11	8	PF11 key value

Variable name	*Length*	*Description*
$PF12	8	PF12 key value
$PF13	8	PF13 key value
$PF14	8	PF14 key value
$PF15	8	PF15 key value
$PF16	8	PF16 key value
$PF17	8	PF17 key value
$PF18	8	PF18 key value
$PF19	8	PF19 key value
$PF20	8	PF20 key value
$PF21	8	PF21 key value
$PF22	8	PF22 key value
$PF23	8	PF23 key value
$PF24	8	PF24 key value

Option 0.3, Job/File Tailoring Parameters

Variable name	*Length*	*Description*
$JFTJESN	8	Initial JES JFT panel name
$JFTMSG1	44	JFT message library concatenation - dataset 1
$JFTMSG2	44	JFT message library concatenation - dataset 2
$JFTMSG3	44	JFT message library concatenation - dataset 3
$JFTMSG4	44	JFT message library concatenation - dataset 4
$JFTMSG5	44	JFT message library concatenation - dataset 5
$JFTPAN1	44	JFT panel library concatenation - dataset 1
$JFTPAN2	44	JFT panel library concatenation - dataset 2
$JFTPAN3	44	JFT panel library concatenation - dataset 3

Variable name	*Length*	*Description*
$JFTPAN4	44	JFT panel library concatenation - dataset 4
$JFTPAN5	44	JFT panel library concatenation - dataset 5
$JFTPEXE	1	Execute any panel authorization flag
$JFTPNME	8	Initial JFT panel name
$JFTSKL1	44	JFT skeleton library concatenation - dataset 1
$JFTSKL2	44	JFT skeleton library concatenation - dataset 2
$JFTSKL3	44	JFT skeleton library concatenation - dataset 3
$JFTSKL4	44	JFT skeleton library concatenation - dataset 4
$JFTSKL5	44	JFT skeleton library concatenation - dataset 5

Option 0.4, GET/PUT TS Queue Identifiers

Variable name	*Length*	*Description*
$TSQSYOV	1	Queue and SYSID override retention
$TSQUEID	8	GET/PUT queue ID save area
$TSSYSID	4	GET/PUT system ID save area

Option 0.5, Utility Parameters

Variable name	*Length*	*Description*
$PRTCLAS	1	Print class
$PRTJCD1	72	Print JOB card 1
$PRTJCD2	72	Print JOB card 2
$PRTJCD3	72	Print JOB card 3
$PRTJCD4	72	Print JOB card 4
$PRTLPGE	3	Print lines per page

Option U, User File Maintenance

Variable name	*Length*	*Description*
$ADDR1	29	Address line 1
$ADDR2	29	Address line 2
$ADDR3	29	Address line 3
$ADDR4	29	Address line 4
$ADMIN	11	Admin authority
$CONTROL	11	Control authority
$DEPART	29	Department
$JESJSCA	4	List of classes the user is authorized to view (4, 1-byte fields)
$JESJSCB	4	Second list of classes the user is authorized to view (4, 1-byte fields)
$JESJSDA	40	List of destinations the user is authorized to view (4, 10-byte fields)
$JESJSDB	40	Second list of destinations the user is authorized to view (4, 10-byte fields)
$JESJSJA	32	List of jobs the user is authorized to view (4, 8-byte fields)
$JESJSMA	4	Miscellaneous field available for user definition and use
$JESJSVA	1	JES2 queue view authority
$NAME	29	User's name
$PRTCLMK	8	CPMS printer selection mask (8, 1-byte fields)
$PRTPRMK	10	CPMS destination selection mask
$PRTPRTA	64	List of printers the user is authorized to view (8, 8-byte fields)
$PRTPRTS	1	If the user can change the criteria the hot writers/JOE writers use to select jobs
$PRTPRTV	1	Printer view authorization

Variable name	*Length*	*Description*
$PSWD	9	User's password
$RECEIVE	11	Receive authority
$TITLE	29	User's title
$USER	11	User ID

Options 1, 2, and 3 (Edit, Browse, and Utility)

Name	*Length*	*Description*
$BRWBOTH	56	Edit other dataset name
$BRWSLB1	8	Browse library 1
$BRWSLB2	8	Browse library 2
$BRWSLB3	8	Browse library 3
$BRWSLB4	8	Browse library 4
$BRWSPRJ	8	Browse project
$BRWSTYP	8	Browse library type
$EDITLB1	8	Edit library 1
$EDITLB2	8	Edit library 2
$EDITLB3	8	Edit library 3
$EDITLB4	8	Edit library 4
$EDITOTH	56	Edit other dataset name
$EDITPRJ	8	Edit project
$EDITTYP	8	Edit library type
$UTILLB1	8	Utility library 1
$UTILLB2	8	Utility library 2
$UTILLB3	8	Utility library 3
$UTILLB4	8	Utility library 4

Name	Length	Description
$UTILPRJ	8	Utility project
$UTILTYP	8	Utility library type

Appendix B
Sample Panels

This appendix provides samples of some JFT panels and the code that generates them.

Note

H&W is interested in seeing any additional examples you develop. Please submit your samples to H&W for possible inclusion in our reference files.

Primary Menu

The following panel shows an example of a primary menu for system utility functions:

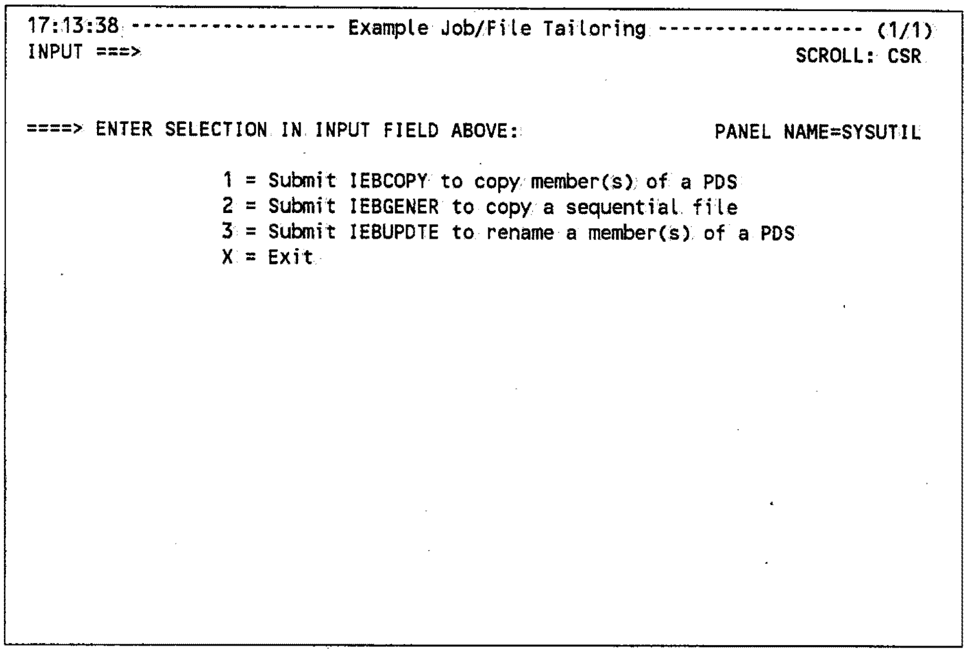
```
17:08:31 ----------------- Example Job/File Tailoring ----------------- (1/1)
INPUT ===>                                                         SCROLL: CSR


====> Enter Selection in Input Field Above:              Panel Name = SYSD08

                 1 = SYSTEM Utilities
                 X = EXIT








 |------------------------------------------------------------------------------|
 |                                                                              |
 | SYSD Option 0.3 sets the Default Panel to Execute upon entry to JFT.         |
 | If you desire your own Main Menu...Update 0.3 with the Name of the           |
 | Panel you wish to execute.                                                   |
 |                                                                              |
 |------------------------------------------------------------------------------|
```

The following code generates this panel:

```
)ATTR DEFAULT(%+_)
 ¬ TYPE(TEXT) INTEN(LOW)
)BODY

¬====> Enter Selection in Input Field Above:%              ¬Panel Name = SYSD08

                 %1¬= SYSTEM Utilities%
                 %X¬= EXIT%








¬|------------------------------------------------------------------------------|
¬|                                                                              |
¬|%SYSD¬Option 0.3 sets the Default Panel to Execute upon entry to%JFT.¬        |
¬| If you desire your own Main Menu...Update 0.3 with the Name of the           |
¬| Panel you wish to execute.                                                   |
¬|                                                                              |
¬|------------------------------------------------------------------------------|
)INIT
 .SYSDTTL = 'Example Job/File Tailoring'
)PROC
   &ZSEL=TRANS(TRUNC(&ZCMD,1),1,'PANEL(SYSUTIL)',
         X,'EXIT',
         ' ',' ',
         *,'?')
)END
```

System Utilities Panel

To view a menu listing the available system utilities, select Option 1 on the primary menu shown on page 70. The following is an example of the System Utilities panel:



```
17:13:38 ------------------ Example Job/File Tailoring ------------------ (1/1)
INPUT ===>                                                          SCROLL: CSR


====> ENTER SELECTION IN INPUT FIELD ABOVE:                   PANEL NAME=SYSUTIL

                 1 = Submit IEBCOPY to copy member(s) of a PDS
                 2 = Submit IEBGENER to copy a sequential file
                 3 = Submit IEBUPDTE to rename a member(s) of a PDS
                 X = Exit
```

The following code generates this panel:

```
)ATTR DEFAULT(%+_)
 ¬ TYPE(TEXT) INTEN(LOW)
)BODY

¬====> ENTER SELECTION IN INPUT FIELD ABOVE:%               ¬PANEL NAME=SYSUTIL

                 %1¬= Submit%IEBCOPY¬to copy member(s) of a PDS%
                 %2¬= Submit%IEBGENER¬to copy a sequential file%
                 %3¬= Submit%IEBUPDTE¬to rename a member(s) of a PDS%
                 %X¬= Exit%
)PROC
   &ZSEL=TRANS(TRUNC(&ZCMD,1),
         1,'PANEL(IEBCOPY)',
         2,'PANEL(IEBGENER)',
         3,'PANEL(IEBUPDTE)',
         X,'EXIT',
         ' ',' ',
         *,'?')
)END
```

Submit IEBCOPY Panel

To execute the IEBCOPY utility, select option 1 on the System Utilities panel shown on page 72. The following is an example of the Submit IEBCOPY panel:

```
17:17:53 ----------------------- Submit IEBCOPY ----------------------- (1/1)
INPUT ===>                                                           SCROLL: CSR

              Jobname: COPY
            Job Class: A          (Must be an 'A' or '8')
             Msgclass: A          (Must be an 'A' or '8')
               Typrun: HOLD       (Must be 'RUN', 'HOLD', or 'SCAN')

        Input Dataset:
                 Disp: SHR
       Output Dataset:
                 Disp: SHR

      Members to Copy:
                       -------- -------- -------- -------- --------
                     :
                       -------- -------- -------- -------- --------
   Replace Member(s): N          (Enter 'Y' to Replace Members)
               Submit: N          (Enter 'Y' to SUBMIT Job)
```

The user enters the data on the panel. The subroutine verifies the fields, reads a skeleton file, substitutes the variables from the panel, and submits the job to the internal reader. The following code generates this panel:

```
)ATTR DEFAULT(%+_)
 ¬ TYPE(TEXT) INTEN(LOW)
 * TYPE(INPUT) INTEN(LOW)
 | TYPE(OUTPUT) INTEN(LOW)
)BODY
              ¬Jobname:*JOBNAM¬
            ¬Job Class:*C¬          (Must be an 'A' or '8')%
             ¬Msgclass:*M¬          (Must be an 'A' or '8')%
               ¬Typrun:*TYPR¬       (Must be 'RUN', 'HOLD', or 'SCAN')%

        ¬Input Dataset:*IDSN                                        %
                 ¬Disp:*IDSP%
       ¬Output Dataset:*ODSN                                        %
                 ¬Disp:*ODSP%

      ¬Members to Copy:*M01     *M02     *M03     *M04     *M05     %
                      ¬-------- -------- -------- -------- --------
                     ¬:*M06     *M07     *M08     *M09     *M10     %
                      ¬-------- -------- -------- -------- --------
    ¬Replace Member(s):*R¬          (Enter 'Y' to Replace Members)%
               ¬Submit:*S¬          (Enter 'Y' to SUBMIT Job)%
```

(*continued*)

```
)INIT
 .SYSDTTL = 'Submit IEBCOPY'          /* set title line */
 &JOBNAM = COPY                       /* set defaults for variables */
 &C = A
 &M = A
 &TYPR = HOLD
 &IDSP = SHR
 &ODSP = SHR
 &R = N
 &S = N
)PROC
 &OM01 = &Z                           /* initialize to nulls */
 &OM02 = &Z
 &OM03 = &Z
 &OM04 = &Z
 &OM05 = &Z
 &OM06 = &Z
 &OM07 = &Z
 &OM08 = &Z
 &OM09 = &Z
 &OM10 = &Z
 VER(&JOBNAM,NB)                      /* jobname must be entered */
 VER(&C,NB,LIST,A,8)                  /* class must be an 'A' or '8' */
 VER(&M,NB,LIST,A,8)                  /* msgclass must be an 'A' or '8' */
 VER(&TYPR,LIST,'RUN ',HOLD,SCAN)     /* typrun must be run, hold, or scan
 VER(&IDSN,NB,DSN)                    /* verify i/o dataset names
 VER(&ODSN,NB,DSN)                    /* are valid */
  IF (&IDSN = 'J')                    /* create short dataset names
   &IDSN = '&$USER..JCL.SOURCE'       /* like DSN003 */
  IF (&ODSN = 'J')
   &ODSN = '&$USER..JCL.SOURCE'
  IF (&IDSN = 'DAS')
   &IDSN = 'SYSD.PTFA.SOURCE'
  IF (&ODSN = 'DAS')
   &ODSN = 'SYSD.PTFA.SOURCE'
  IF (&IDSN = 'DBS')
   &IDSN = 'SYSD.PTFB.SOURCE'
  IF (&ODSN = 'DBS')
   &ODSN = 'SYSD.PTFB.SOURCE'
  IF (&IDSN = 'DBL')
   &IDSN = 'SYSD.PTFBMVS.LOADLIB'
  IF (&ODSN = 'DBL')
   &ODSN = 'SYSD.PTFBMVS.LOADLIB'
  VER(&M01,NAME)                      /* verify all member names
  VER(&M02,NAME)                      /* are valid */
  VER(&M03,NAME)
  VER(&M04,NAME)
  VER(&M05,NAME)
  VER(&M06,NAME)
  VER(&M07,NAME)
  VER(&M08,NAME)
  VER(&M09,NAME)
  VER(&M10,NAME)
  VER(&R,NB,LIST,Y,N)                 /* replace option must be Y or N */
  IF(&M01 ¬= &Z) &OM01 = 'S M=&M01' /* attach 's m=' to the front
  IF(&M02 ¬= &Z) &OM02 = 'S M=&M02' /* of each member */
  IF(&M03 ¬= &Z) &OM03 = 'S M=&M03'
  IF(&M04 ¬= &Z) &OM04 = 'S M=&M04'
  IF(&M05 ¬= &Z) &OM05 = 'S M=&M05'
  IF(&M06 ¬= &Z) &OM06 = 'S M=&M06'
  IF(&M07 ¬= &Z) &OM07 = 'S M=&M07'
  IF(&M08 ¬= &Z) &OM08 = 'S M=&M08'
  IF(&M09 ¬= &Z) &OM09 = 'S M=&M09'
  IF(&M10 ¬= &Z) &OM10 = 'S M=&M10'
  IF(&R = N)                        /* build NO REPLACE control stm */
   &CONTROL = 'COPY INDD=IN01,OUTDD=OUT01'
  IF(&R = Y)                        /* build REPLACE control stm */
   &CONTROL = 'COPY INDD=((IN01,R)),OUTDD=OUT01'
 IF(&S = Y)                         /* submit the job when ready
  IF(.MSG = &Z)                     /* don't submit if errors
   SUBSKEL(IEBCOPY)                 /* using the skeleton member
  &S=N                              /* IEBCOPY and turn submit flag off*/
)END
```

The following is an example of the skeleton file used for the IEBCOPY panel:

```
//&$USER&JOBNAM JOB '&$NAME',
//              MSGCLASS=&M,CLASS=&C,TYPRUN=&TYPR
/*ROUTE PRINT &$DESTID
//IEBCOPY EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//IN01 DD DISP=&IDSP,DSN=&IDSN
//OUT01 DD DISP=&ODSP,DSN=&ODSN
//SYSIN DD *
 &CONTROL
 &OM01
 &OM02
 &OM03
 &OM04
 &OM05
 &OM06
 &OM07
 &OM08
 &OM09
 &OM10
/*
//
```

Appendix C
JFTADD Program

This appendix provides the complete code for the JFTADD program.

```
*ASM CICS(NOPROLOG NOEPILOG)          COMMAND LEVEL TRANSLATOR
         COPY  SYSDGBLS
         SETSSI
         LINKCTL COMMAND
&PGMNAME SETC  'JFTADD'
         ATITLE 'JFTADD - EXEC CICS LINK JFT EXAMPLE',                *
               TYPE=TOPIC
*PROGRAM
*              JFTADD - EXEC CICS LINK JFT EXAMPLE
*COPYRIGHT
*              (C)COPYRIGHT 1997, H & W COMPUTER SYSTEMS, INC.
*PROGRAMMER
*              PETER J. PAPE
*DATE WRITTEN
*              AUGUST 21, 1997
*PURPOSE
*              THIS PROGRAM ADDS TWO NUMBERS ENTERED FROM A JFT
*              PANEL.  THE PANEL ASSIGN AND VERIFY JFT VARIABLE
*              NAMES NUM1 AND NUM2.  NUM1 AND NUM2 MUST BE NUMERIC.
*
*              THE $PROC AND $END ARE REQUIRED.  THE FIRST $PROC MUST
*              HAVE OPTIONS OF MAIN AND COMMAND.  BY SPECIFYING GEN,
*              ALL MACROS WILL BE EXPANDED.  ADDITIONAL $PROC AND $END
*              MACROS MAY BE USED AND INVOKED WITH A $CALL MACRO.
*              EACH $PROC AND $END ARE THE BEGINNING AND ENDING OF A
*              CSECT.  THE $PROC THAT HAS THE OPTIONS OF MAIN IS THE
*              CSECT THAT GETS CONTROL WHEN LINKED TO.
*
*              FOR FURTHER EXAMPLES, REFER TO SYSD.SOURCE.  IF THE
*              MACRO EXPANSIONS ARE NECESSARY, OPTION 'LIST' MUST
*              BE USED WHEN ASSEMBLING.
*
*MAINTENANCE LOG
         MODID
*        PUT   06.4.0
*              PJ 08/21/97 CREATED.
*INPUT
*              VARIABLES FROM THE FUNCTION POOL.
```

```
*OUTPUT
*              UPDATED VARIABLES FROM THE FUNCTION POOL.
*OS/VS MACROS USED
         PRINT OFF
         COPY DFHCSADS
         COPY DFHTCADS
         DFHBFTCA OPTION=BASIC
         PRINT ON
         SYSDTWA
TWAEND   EQU   *                   END OF TWA
TWALGHT  EQU   TWAEND-TWASTART     LENGTH OF TWA
         ATITLE '** COMMAND LEVEL DYNAMIC STORAGE **'
***********************************************************************
*                COMMAND LEVEL DYNAMIC STORAGE                        *
***********************************************************************
*               AREA HEADER INFORMATION.
DFHEISTG DSECT
         DFHEISTG ,
COMMBEGN EQU   *
*** PLACE DYNAMIC STORAGE AREAS HERE.
EBACNUM1 DS    CL9                 FIRST NUMBER TO ADD
NUM1LEN  EQU   L'EBACNUM1          LENGTH OF NUMBER 1
EBACNUM2 DS    CL9                 SECOND NUMBER TO ADD
NUM2LEN  EQU   L'EBACNUM2          LENGTH OF NUMBER 2
EBACSUM  DS    CL11                RETURN THE NUMBER BACK.
*              AREA TRAILER INFORMATION.
COMMLGTH EQU   *-COMMBEGN          LENGTH OF AREA
         ATITLE '** EQUATES **'
         REGEQU
SAVER    EQU   R8                  REGISTER SAVE AREA POINTER
BASER    EQU   R9                  PROGRAM BASE REGISTER
         ATITLE ' ** PROGRAM MAIN PROCEDURE **'
***********************************************************************
*                     PROGRAM MAIN PROCEDURE                          *
***********************************************************************
&PGMNAME $PROC OPTIONS=(MAIN,COMMAND,GEN)
         $DCL  ON
WPFLD1   DS    PL5                 PACKED WORK AREA FOR NUMBER 1
WPFLD2   DS    PL5                 PACKED WORK AREA FOR NUMBER 2
WPSUM    DS    PL6                 PACKED WORK AREA FOR NUMBER SUM
         $DCL OFF
*
*               GET THE NUMBERS TO ADD
*
         $CMVGET ('NUM1     ',9,EBACNUM1)    FIRST NUMBER
         $CMVGET ('NUM2     ',9,EBACNUM2)    SECOND NUMBER
*
*               CONVERT THE DATA TO PACK NUMBERS
*
         $CALL V(VALNUM),(EBACNUM1,NUM1LEN)
         PACK  WPFLD1,EBACNUM1         PACK THE FIRST NUMBER
*
         $CALL V(VALNUM),(EBACNUM2,NUM2LEN)
         PACK  WPFLD2,EBACNUM2         PACK THE SECOND NUMBER
*
*               NOW WE ADD THEM
*
         ZAP   WPSUM,WPFLD1            MOVE FIRST NUMBER TO THE SUM FLD
         AP    WPSUM,WPFLD2            ADD SECOND NUMBER TO THE FIRST
```

```
*
*                CONVERT TO DISPLAY
*
         UNPK  EBACNUM1,WPFLD1          UNPACK THE NUMBER
         OI    EBACNUM1+(L'EBACNUM1-1),X'F0'
         UNPK  EBACNUM2,WPFLD2          UNPACK THE NUMBER
         OI    EBACNUM2+(L'EBACNUM2-1),X'F0'
         UNPK  EBACSUM,WPSUM            UNPACK THE NUMBER
         OI    EBACSUM+(L'EBACSUM-1),X'F0'
*
*                PASS THE NUMBERS BACK TO JFT
*
         $CMVPUT ('NUM1     ',9,EBACNUM1,'F') PASS NUMBER1 TO JFT PANEL
         $CMVPUT ('NUM2     ',9,EBACNUM2,'F') PASS NUMBER2 TO JFT PANEL
         $CMVPUT ('SUM      ',11,EBACSUM,'F') PASS SUM TO JFT PANEL
         $END  RC=0
         ATITLE ' VALIDATE NUMERIC FIELD '
***********************************************************************
*    VALNUM - VALIDATE NUMERIC FIELD                                  *
*                                                                     *
*        THIS ROUTINE TAKES THE INPUT NUMBER AND LENGTH FIELD AND     *
*        RIGHT JUSTIFIES THE NUMBER AND FILLS ZEROS TO THE LEFT       *
*        TO MAKE THE NUMBER VALID FOR PACK INSTRUCTION.               *
*                                                                     *
*    INPUT                                                            *
*        CHARACTER FIELD TO BE CONVERTED TO NUMERIC DATA              *
*        LENGTH OF THE FIELD TO BE CONVERTED                          *
*    OUTPUT                                                           *
*        RETURNS ZERO - UPDATES MEMORY LOCATION OF VALUE PASSED       *
*                                                                     *
***********************************************************************
VALNUM   $PROC
         $DCL  NUMBEGN,FUL              POINTER TO PASSED VALUE
         $DCL  NUMLEN,FUL               LENGTH OF THE FIELD PASSED
         $DCL  PADLEN,FUL               LENGTH OF THE FIELD PASSED
         $DCL  WORKNUM,CHAR(10)         WORK FIELD
         LM    R2,R3,0(R1)              GET THE PARM DATA
         LA    R3,0(,R3)                CLEAR THE HIGH ORDER BIT
         ST    R2,NUMBEGN               SAVE POINTER TO VALUE PASSED
         ST    R3,NUMLEN                SAVE LENGTH OF VALUE PASSED
*
*        DETERMINE HOW MANY ZEROS NEED TO BE INSERTED
*
VALNUM10 DS    0H                       WHILE R3 > ZERO
         C     R3,=X'00000000'            IF R3 > ZERO
         BL    VALNUM20                     STOP THE LOOP
         CLI   0(R2),X'40'                IF CHAR = SPACES
         BE    VALNUM20                     STOP THE LOOP
*                                          ELSE
         LA    R2,1(,R2)                    POINT TO THE NEXT CHARACTER
         BCT   R3,VALNUM10                  DECREMENT LENGTH COUNTER(R3)
VALNUM20 DS    0H                       ENDWHILE
         ST    R3,PADLEN                SAVE NUMBER OF BYTES TO PAD
```

```
*
*        ZERO PAD AND RIGHT JUSTIFY
*
         L     R2,NUMBEGN                   POINT TO THE INPUT NUMBER
         LA    R3,0                         SET R3 TO ZERO
         LA    R4,WORKNUM                   POINT TO THE WORK NUMBER
VALNUM30 DS    0H
         C     R3,NUMLEN                    WHILE R3 < LENGTH OF NUM FLD
         BL    VALNUM35                       CONTINUE
         B     VALNUM60                       ELSE STOP THE LOOP
VALNUM35 DS    0H
         C     R3,PADLEN                      IF R3 > OR EQUAL PAD LENGTH
         BL    VALNUM40
         MVC   0(1,R4),0(R2)                    MOVE THE NUMBER IN
         LA    R2,1(,R2)                        POINT TO NEXT INPUT BYTE
         B     VALNUM50
VALNUM40 DS    0H                             ELSE
         MVI   0(R4),X'F0'                      MOVE ZERO IN FOR PAD
VALNUM50 DS    0H                             ENDIF
         LA    R4,1(,R4)                      POINT TO NEXT OUTPUT BYTE
         LA    R3,1(,R3)                      ADD ONE TO THE NUMBER OF BYTES
         B     VALNUM30
VALNUM60 DS    0H                           ENDWHILE
*
*        MOVE THE DATA TO THE WORK FIELD TO THE PASSED FIELD
*
         L     R2,NUMBEGN                   POINT TO FIRST BYTE
         L     R1,NUMLEN                    GET LENGTH TO MOVE
         BCTR  R1,0                         DECREASE LENGTH BY 1 FOR MVC
         EX    R1,MOVEDATA                  GO DO THE MOVE
         $END  RC=0
MOVEDATA MVC   0(*-*,R2),WORKNUM
         ATITLE ' ** AUTOMATIC SUBROUTINE GENERATION **'
***********************************************************************
*                    AUTOMATIC SUBROUTINE GENERATION                  *
***********************************************************************
         $SBRTNG
         ATITLE ' ** AUTOMATIC DSECT GENERATION **'
***********************************************************************
*                    AUTOMATIC DSECT GENERATION                       *
***********************************************************************
         $DSECTG
         ATITLE ' ** COMMAND LEVEL CLEAN UP **'
***********************************************************************
*                     COMMAND LEVEL CLEAN UP                          *
***********************************************************************
&PGMNAME CSECT
         PRINT GEN
         DFHEISTG
         DFHEIEND
         END
```

Index

!

$CALL macro 60
$CMVGET macro 54
$CMVPUT macro 55
$DCL macro 58-59
$END macro 58-59
$PROC macro 58-59
$RETURN macro 58
&$DEPART profile variable 22
&$DESTID profile variable 22
&$NAME profile variable 22
&$TITLE profile variable 22
&$USER profile variable 22
&Z system variable 20
&ZCMD system variable 20
&ZCSR system variable 20
&ZDATE system variable 20
&ZDAY system variable 20
&ZJDATE system variable 20
&ZJ4DATE system variable 21
&ZJDATEST system variable 20
&ZMONTH system variable 21
&ZSEL system variable 21
&ZSTD system variable 21
&ZTIME system variable 21
&ZYEAR system variable 21
)ATTR section 8-9
)BLANK control statement 48
)BODY section 10-11
  variables 19-25
)CM control statement 48
)DEFAULT control statement 49
)END section 17
)IM control statement 49
)INIT section 12-13
  logic statements 27-45
  variables 19-25
)PROC section 14-16
  invoking new panel after processing 21
  logic statements 27-45
  variables 19-25
)TB control statement 49
.CURSOR control variable 23
.CURSORF control variable 23
.MSG control variable 23-24
.MSGF control variable 24
.SYSDTTL control variable 24
.TRAIL control variable 24

A

Accessing
  system information 20-21
  variables from CICS programs 54
Allocating data areas 58-59
Assigning values to variables 29-31, 38-39
Assignment statement 29-31
  comparing variables to values 38-39
  concatenation rules 29-30
  defining title line 24
  invoking new panel after processing)PROC section 21
Attribute characters
  defaults 9
  defining 8-9
  overriding defaults 9

B

Blank lines
)BODY section 10
  inserting in output skeleton files 48
Building parameter lists 60

C

Calling CICS programs 53-60
  example of 56-57
Changing
  default control statement character 49
  library concatenations dynamically 4
Checking return codes from subroutines 58
CICS programs
  calling 53-60
  example of calling 56-57
  linking to 36
  reading variables into storage 54
  updating variables from 55
Comments
)BODY section 10
  logic statements 28
  skeletons 48
Comparing variables
  to values 38-39
  using &Z system variable 20
Components 2-3
Concatenating panel, skeleton, and message libraries 4
Concatenation rules, assignment statement 29-30
Conditionally executing statements 34-35
Constant data 58
Continuing
  parameters on logic statements 28
  quoted strings on logic statements 28
Control statement character, changing default 49
Control statements 48-51
)BLANK 48
)CM 48
)DEFAULT 49
)IM 49
)TB 49-51
Control variables 23-24
Conventions, syntax ix
Copying members to output skeleton files 49
Cursor position
  getting for *Input* field 20
  overriding previous 23
Cursor, positioning in fields 23
Customizing
  messages 3
  panels for groups of users 2

D

Data
  constant 58
  getting from *Input* field 20
  positioning in output skeleton files 49-51
  storing in JFT-like programs 58-59
Data areas, allocating 58-59
Date, system 20-21
Default attribute characters 9
  overriding 9
Default control statement character, changing 49
Default messages, overriding 3
Default panel 2
Defining
  attribute characters 8-9
  panel appearance 10-11
  user storage areas 58-59
Department name, getting from user file 22
Destination, getting from Option 0.1 22
Dynamic Storage Area, defining global variables 59
Dynamically changing library concatenations 4

E

Evaluating expressions 29-31, 34-35
Executing panels 6
Expressions
  assigning to variables 29-31
  evaluating 29-31, 34-35

F

Fields, panel
  intensity 8-9
  positioning cursor in 23
  type 8
Files
  message 52
  skeleton 48-51
FILESKEL statement 32-33
  substituting variables in output skeleton files 48-51

G

Global variables 59

I

IEBCOPY utility, sample panel 73-75
IF statement 34-35

using &Z system variable for compare 20
Initializing
panels 12-13
variables using &Z system variable 20
Input data, getting from *Input* field 20
Input field
getting cursor position of 20
getting data from 20
Input fields, defining 8
Inserting blank lines in output skeleton files 48
Intensity of panel fields, defining 8-9
Internal reader, submitting skeletons to 37
Invoking new panel after processing)PROC section 21

J

JFT-like programs
defining user storage areas 58-59
storing data in 58-59
JFTADD program
complete code 77-80
example of calling 56-57
example of getting variables 59-60
Julian date 20

K

Keyboard conventions ix

L

Libraries
concatenating 4
copying concatenations 49
dynamically changing concatenations 4
LINK statement 36
Linking to CICS programs 36
Local variables 58-59
Logic statements 27-45
assignment 29-31
comments 28
continuing parameters 28
continuing quoted strings 28
FILESKEL 32-33
IF 34-35
LINK 36
SUBSKEL 37
syntax rules 28
TRANS 38-39
TRUNC 40-41
VER 42-45

M

Macros
$CALL 60
$CMVGET 54
$CMVPUT 55
$DCL 58-59
$END 58-59
$PROC 58-59
$RETURN 58
Manual organization viii
Marking
beginning of programs and subroutines 58-59
end of programs and subroutines 58-59
Members, saving using skeletons 32-33
Menu system variables 61-67
Messages
concatenating libraries 4
customizing 3
definition 3
displaying on panels 52
dynamically changing library concatenations 4
files 52
overriding default 3
overriding previous 24
setting 23-24
storing 4

N

Name, getting from user file 22

O

Organization of manual viii
Output fields, defining 8
Output skeleton files
copying members to 49
inserting blank lines 48
setting tabs in 49-51
substituting variables in 48
Overriding
default attribute characters 9
default messages 3
previous cursor positioning 23

previous message 24
Overview 1-6

P

Panel sections 7-17
)ATTR 8-9
)BODY 10-11
)END 17
)INIT 12-13
)PROC 14-16
Panels
appearance 10-11
attribute characters 8-9
calling CICS programs 53-60
concatenating libraries 4
customizing for groups of users 2
default 2
default attribute characters 9
defining title line 24
definition 2
displaying messages 52
dynamically changing library concatenations 4
executing 6
field intensity 8-9
field types 8
initializing 12-13
invoking new after processing)PROC section 21
linking to CICS programs 36
logic statements 27-45
message files 52
overriding default attribute characters 9
processing 14-16
samples 69-75
skeleton files 48-51
storing 4
testing 6
variables 19-25
Parameters
building lists 60
continuing on logic statements 28
using Register 1 to pass 58
Passing parameters using Register 1 58
Position of cursor
getting for *Input* field 20
overriding previous 23
Positioning
cursor in fields 23
data in output skeleton files 49-51
Primary menu, sample panel 70-71
Processing panels 14-16
Profile variables 22
Programs, CICS
calling 53-60
example of calling 56-57
reading variables into storage 54
updating variables from 55
Programs, JFT-like
defining user storage areas 58-59
storing data in 58-59
Programs, SYSD-like 58-60
checking return codes 58
marking beginning of 58-59
marking end of 58-59
restoring registers at end 58-59
saving registers 58-59
setting return codes 58
Protected text fields, defining 8
Publications, related x

Q

Quoted strings, continuing on logic statements 28

R

Reading variables into storage 54
Register 1
passing parameters 58
pointing to parameter lists 60
Register 15, checking 58
Registers
restoring at end of programs and subroutines 58-59
saving from calling programs 58-59
Related publications x
Resetting
cursor positioning 23
messages 24
Restoring registers at end of programs and subroutines 58-59
Return codes, setting in subroutines 58

S

Sample panels 69-75
Primary Menu 70-71
Submit IEBCOPY 73-75
System Utilities 72
Saving
members using skeleton 32-33
registers from calling programs 58-59

Setting
  cursor position to fields 23
  messages 23-24
  return codes in subroutines 58
  tabs for output skeleton files 49-51
Skeleton output files
  copying members to 49
  inserting blank lines in 48
  setting tabs for 49-51
  substituting variables in 48
Skeletons 48-51
  changing default control character 49
  comments 48
  concatenating libraries 4
  control statements 48-51
  definition 2
  dynamically changing library concatenations 4
  storing 4
  submitting to internal reader 37
  using to save members 32-33
Storage areas, defining user 58-59
Storing
  data in JFT-like programs 58-59
  messages 4
  panels 4
  skeletons 4
Strings, continuing quoted 28
Submit IEBCOPY panel, sample 73-75
Submitting skeletons to internal reader 37
Subroutines
  checking return codes 58
  marking beginning of 58-59
  marking end of 58-59
  restoring registers at end 58-59
  setting return codes 58
SUBSKEL statement 37
  substituting variables in output skeleton files 48-51
Symbol conventions ix
Syntax conventions ix
Syntax rules
)BODY section 10
  logic statements 28
SYSD user file, getting information from 22
SYSD-like programs 58-60
  checking return codes 58
  marking beginning of 58-59
  marking end of 58-59
  restoring registers at end 58-59
  saving registers 58-59
  setting return codes 58
System date 20-21
System information, accessing 20-21
System time 21
System Utilities panel, sample 72
System variables 20-21
SYSUTIL panel, sample 72

T

Tabs for skeleton output files, setting 49-51
Testing panels 6
Text conventions ix
Text fields, defining 8
Time, system 21
Title line, panel 24
Title, getting from user file 22
TRANS statement 38-39
TRUNC statement 40-41
Truncating variables 40-41
  assigning leftover characters to .TRAIL 24

U

Updating variables from CICS programs 55
User file, getting information from 22
User ID, getting from user file 22
User storage areas, defining 58-59
User variables 25

V

Variables 19-25
  accessing from CICS programs 54
  assigning values 29-31, 38-39
  comparing to values 38-39
  control 23-24
  global 59
  local 58-59
  menu system 61-67
  overview 5
  profile 22
  reading into storage from CICS programs 54
  skeleton files 48
  substituting in message files 52
  substituting in output skeleton files 48
  system 20-21
  truncating 40-41
  updating from CICS programs 55
  user 25
  using &Z system variable to compare 20
  using &Z system variable to initialize 20
  verifying values 42-45
VER statement 42-45

positioning cursor in fields 23
setting messages 23-24
Verifying values of variables 42-45