Order No. SC34-2036-1
File No. S370/4300-39

Program Product

System Productivity Facility Dialog Management Services

Program Number 5668-009

IBM

Second Edition (March, 1981)

This edition applies to the System Productivity Facility (SPF) Program Product (5668-009), for use with the following:

OS/VS2 MVS Release 3.8
VM/SP

and to all subsequent releases until otherwise indicated by Technical Newsletters. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370 and 4300 Processors Bibliography,* GC20-0001, for the editions that are applicable and current.

PREFACE

The System Productivity Facility (SPF) is a program product that assists in program development. It is designed to take advantage of the characteristics of IBM 3270 display terminals, and to increase programmer productivity in an interactive environment.

The System Productivity Facility replaces the previous Structured Programming Facility program products (SPF/TSO and SPF/CMS). It includes significant new functions that support the development, testing, and execution of interactive applications.

New services are provided to display predefined screen images and messages, build and maintain tables of user information, generate output files for job submission or other processing, define and control symbolic variables, interface to edit and browse, and log hardcopy output.

This manual provides a detailed description of the new SPF services and related information required to develop an interactive application that runs under SPF. It applies to both the MVS/TSO and VM/CMS environments.

The manual is divided into the following chapters:

1. Introduction - A general overview of SPF, including its structure and function.
2. Concepts and Facilities - A description of the services and control facilities that support the operation of interactive applications.
3. SPF Invocation - A description of the library setup requirements, and the ISPF command used to invoke the dialog manager.
4. Description of Services - A detailed description of each dialog service, including syntax conventions for command or call invocation.
5. Panel, Message, and Skeleton Formats - A detailed description of the syntax for defining panels, messages, and file tailoring skeletons.
6. Dialog Testing Procedures - A description of the test aids available for testing dialogs.

The manual also includes two appendixes:

A. Sample Problem
B. Summary of SPF Dialog Services

TERMINOLOGY

In this manual, the following terms are used to bridge the differences in terminology between the MVS and VM environments:

- Library - A partitioned data set in the MVS environment, or a MACLIB in the VM environment.
- File - A sequential data set in the MVS environment, or a sequential CMS file in the VM environment.
- Command Procedure - A CLIST in the MVS environment, or an EXEC (coded in EXEC2 language) in the VM environment.

NOTATION CONVENTIONS

The following notation conventions are used for describing the ISPF command and each SPF service:

- Command verbs, service names, and keywords that must be coded exactly as shown are represented with uppercase characters.
- Substitutable operands are represented with lowercase characters.
- Optional parameters are enclosed in brackets, "[" and "]".
- Within brackets, a choice of parameters is indicated with slashes and the default is underscored.
- Multiple choice parameters, of which one is required, are enclosed in braces, "{" and "}", and aligned vertically.

Example:

```
ISPEXEC  VGET  name-list  [SHARED/PROFILE]
```

The command verb (ISPEXEC) and the service name (VGET) must be coded exactly as shown. A list of variable names must be substituted for "name-list". The keyword parameter is optional. Either SHARED or PROFILE may be coded. If the parameter is omitted, SHARED is the default.

RELATED DOCUMENTS

This document, SPF Dialog Management Services, applies to both the MVS/TSO and VM/CMS environments. For the other SPF documentation, separate manuals are provided for the two environments, as follows:

SPF-MVS	SPF-VM	DESCRIPTION
GC34-2039	GC34-2046	SPF General Information Provides an overview and functional description of SPF.
SC34-2038	SC34-2047	SPF Program Reference Provides detailed information on how to use the SPF program development facility.
SC34-2037	SC34-2048	SPF Installation and Customization Provides detailed information on how to install and custom tailor SPF.

CONTENTS

This revision to SPF Dialog Management Services includes:

- Additional information pertaining to operation of SPF in the VM/CMS environment.
- Corrections to errors in the previous edition.
- Clarification of information in the previous edition.

All technical changes are marked with a vertical bar (|) on the left-hand side of the page.

CHAPTER 1. INTRODUCTION

The System Productivity Facility (SPF) is a program product that replaces both of the previous Structured Programming Facility products (SPF/TSO and SPF/CMS). SPF supports two environments:

- MVS Time Sharing Option (SPF-MVS)
- VM/SP Conversational Monitor System (SPF-VM)

The new name, System Productivity Facility, reflects the addition of significant new functions beyond the support for structured programming. The new functions support the development, testing, and execution of interactive applications that run under control of SPF and use new SPF services to:

- Display predefined screen images and messages
- Build and maintain tables of user information
- Generate output files for job submission or other processing
- Define and control symbolic variables
- Interface to edit and browse, and log hardcopy output
- Control operational modes.

SPF consists of two major components: the dialog manager and the program development facility (see Figure 1). Conceptually, the dialog manager is an extension to the operating system. It provides control and services for running interactive applications. One such application is the program development facility, which includes the previous SPF functions.

```
DIALOG MANAGER

   CONTROL FACILITIES
      Menu/Tutorial Processing
      Screen Management
      Program Key Interpretation

   SERVICES
      Display
      Table Creation/Maintenance
      File Tailoring
      Variable Definition/Control
      Other

PROGRAM DEVELOPMENT FACILITY

   SPF Parms
   Browse
   Edit
   Utilities
   Foreground
   Background (Batch)
   Command
   Support
   Tutorial
```

Figure 1. SPF Organization

The dialog manager allows totally new applications to be developed. They may be DP-oriented applications, including extensions to the SPF program development facility, or applications for which the end user is not a DP-professional, such as an inventory application.

Displays and messages may be tailored to the particular needs and terminology of the end user, so that information is presented in a user-friendly way.

An installation may have several applications that run under the dialog manager. They may be independent, entered via separate command procedures, or linked via menu options that transfer from one application to another. A sample "master menu," distributed with SPF, allows application selection to occur on entry to SPF. Its use is not required; any selection menu may have options that link to other applications.

Applications may be coded in:

- The command language of the host system (CLIST for MVS/TSO, or EXEC2 for VM/CMS), or

- A programming language, such as PL/I or COBOL.

An application may have mixed languages, with some functions coded in a command language and other functions coded in one or more programming languages.

Applications coded in programming languages may be designed for cross-system use, to be run under either MVS or VM.

SPF also provides new testing aids for debugging interactive applications. These are part of the SPF program development facility -- the SUPPORT option (primary option 7).

CHAPTER 2. CONCEPTS AND FACILITIES

This chapter describes basic concepts of dialog organization and control, and the capabilities of the SPF dialog services.

ELEMENTS OF A DIALOG

A "dialog" is any application designed to be run under the SPF dialog manager. Each dialog is composed of program and data elements, which allow an orderly interaction between the computer and the end user of the application. The types of elements that make up a dialog are:

- Functions. A function is a program that performs processing requested by the user. It may invoke SPF dialog services to display panels and messages, build and maintain tables, and generate output files. A function may be coded in a command language (CLIST or EXEC2) or a programming language.

- Panels.[1] A panel is a predefined display image. It may be a selection menu, a data entry display, or an information-only display. Most panels prompt the user for input. The user response may identify which path is to be taken through the dialog, or it may be interpreted as data.

- Messages. A message is a comment that provides special information to the user. It may confirm that a user-requested action is in progress or completed, or report an error in the user's input. Messages may be directed to the user's terminal, to a hardcopy log, or both.

- Tables. A table is a two-dimensional array used to maintain data. A table may be created as a temporary data repository, or it may be retained across sessions. A retained table may also be shared between different applications. The type and amount of data stored in a table depends upon the nature of the application.

- File Skeletons. A file skeleton is a generalized representation of sequential data which may be customized during dialog execution to produce an output file. The output file may be used to drive other processes. File skeletons are frequently used to produce job files for batch execution.

A dialog need not include all types of elements. In particular, tables and file skeletons may not be needed, depending upon the type of application.

Panels, messages, and file skeletons are stored in libraries prior to execution of the dialog. They are created by editing directly into the panel, message, or skeleton libraries; no compile or preprocessing step is required.

Tables are generated or updated dynamically during dialog execution. The organization of each table is specified to SPF by the functions that use SPF table services.

[1] Previously, all SPF screen images were called menus. The terminology has been changed to more closely reflect general usage. The term menu is now used to mean a display from which the user may select options. The term panel is used to mean any predefined display image, of which one type is a menu.

DIALOG ORGANIZATION

A dialog may be organized in a variety of ways to suit the requirements of the application and the needs of the end user. A typical dialog organization is shown in Figure 2.

This example starts with the display of a high level selection menu, the primary option menu for the application. User options selected from this menu may result in the invocation of a dialog function, or the display of a lower level selection menu. Each lower level menu may also cause functions to receive control, or still lower level menus to be displayed. The menu hierarchy may extend as many levels deep as desired.

Eventually a dialog function will receive control. The function may use any of the dialog services provided by SPF. In particular, the function may continue the interaction with the end user by means of the DISPLAY service. Typically, the function will display data entry panels to prompt the user for information.

When the function completes, the selection menu from which it was invoked is redisplayed.

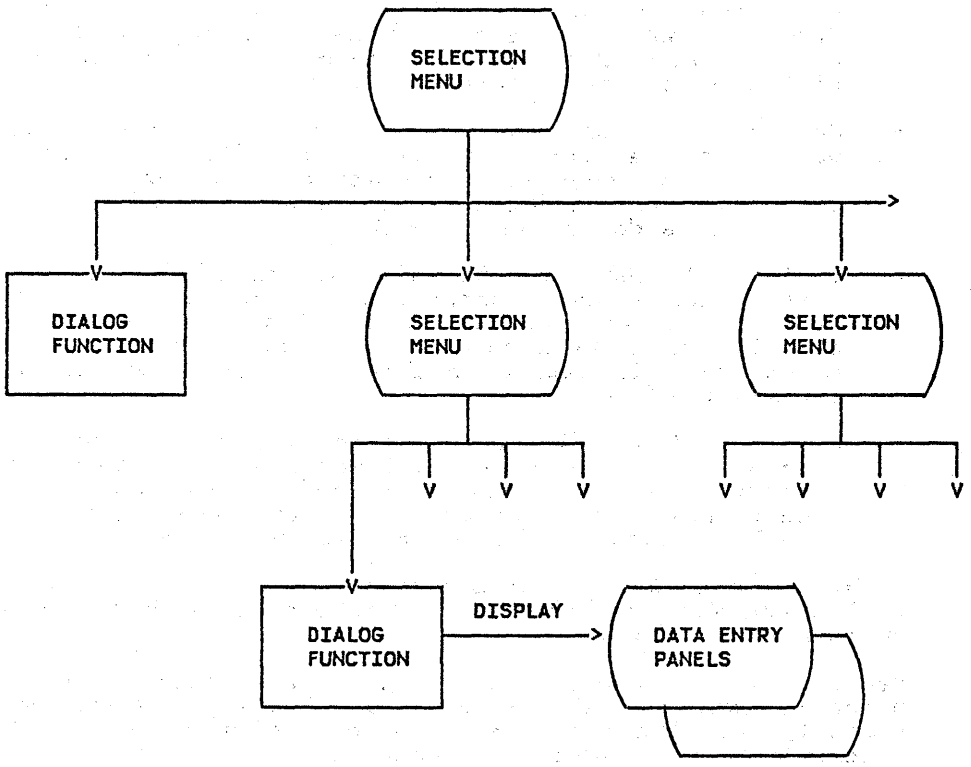


Figure 2. Typical Dialog Organization

Figure 3 shows a different type of dialog organization. In this example, a dialog function receives control first, prior to the display of a menu. It may perform application-dependent initialization, and display data entry panels to prompt the user for basic information. It may then start the selection process by using the SELECT service to display the primary option menu for the application.

This example also shows one function invoking another function, via SELECT, without displaying a menu. This provides a convenient way to pass control from a program-coded function to a command-coded function, or vice versa. The invoked function then starts a lower level selection process, again by using the SELECT service.

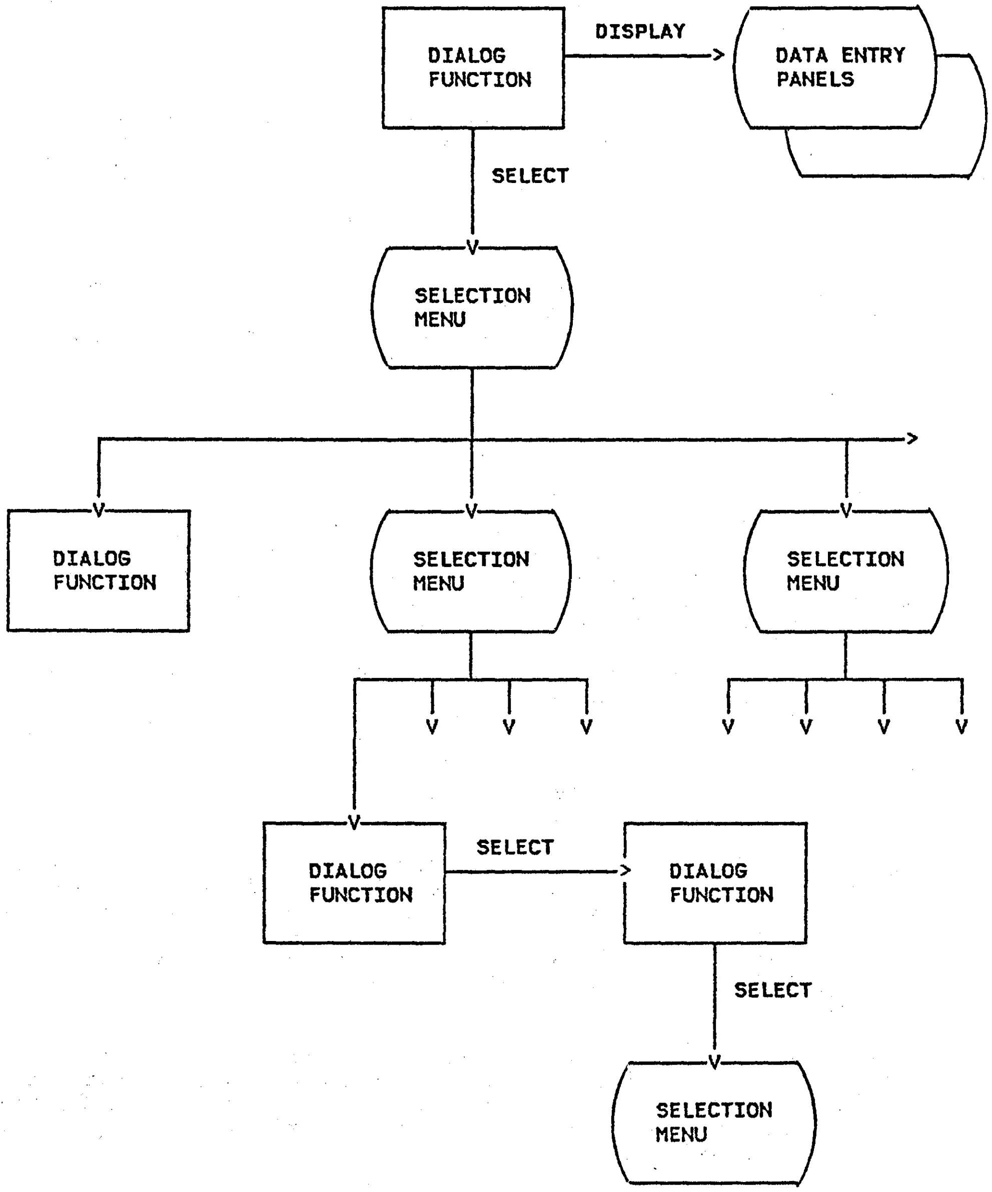


Figure 3. Other Dialog Organizations

FLOW OF CONTROL

The flow of control is shown in Figure 4. A dialog is started by means of the ISPF command. (An SPF command may be established as an alias of ISPF.) The command may be entered:

- By a user at the terminal,
- From a command procedure (CLIST or EXEC2), or
- During LOGON (from a TSO LOGON procedure or CMS PROFILE EXEC).

When the ISPF command is used to invoke the SPF program development facility, no command parameters are required.

When the ISPF command is used to invoke any other dialog, command parameters are used to specify the first selection menu to be displayed or the first dialog function to receive control (prior to the display of a menu). In this case, the ISPF command is typically entered from a command procedure or during LOGON.

Example: The user might invoke a new application by entering the ABC command. ABC would be coded as a command procedure that allocates the appropriate libraries for the application, and then issues an ISPF command with a parameter that specifies the first selection menu or dialog function. The ABC command serves as a "front end" to start the application. It may not use SPF dialog services, since it is not running under SPF.

The ISPF command invokes the SPF controller, which initializes the environment. The controller then calls the SELECT service to display the first menu or invoke the first dialog function, passing it the parameters specified on the ISPF command.

SELECT SERVICE

SELECT is both a control facility and a dialog service. It controls the sequence of selection menus, based on user inputs, and invokes dialog functions. From a dialog function, SELECT may be used as a service to start the display of a menu hierarchy or invoke other functions without displaying a menu.

Selection menus contain sufficient information to determine the next action to be taken for any option entered by the user. This information is interpreted by the SELECT service. The next action may be:

- Display a lower level selection menu, or
- Invoke a dialog function.

When the SELECT service is used to display a menu hierarchy, it will display the first menu (via the DISPLAY service) and continue to display successively lower levels of menus, until a dialog function is specified as the next action. The SELECT service will then invoke the function. When the function completes execution, the SELECT service will redisplay the menu from which the function was invoked.

When the SELECT service is used to invoke a function without displaying a menu, it simply passes control to the lower level function. When that function completes execution, SELECT returns to the function that called it, passing along the return code from the lower level function.

Parameters may be passed to any dialog function from the selection menu or function that invoked it (or from the ISPF command, if it is the first function to receive control prior to the display of a menu). These parameters may be used, for example, to pass the name of a panel to be displayed, a table to be updated, or a file skeleton to be used by the function.

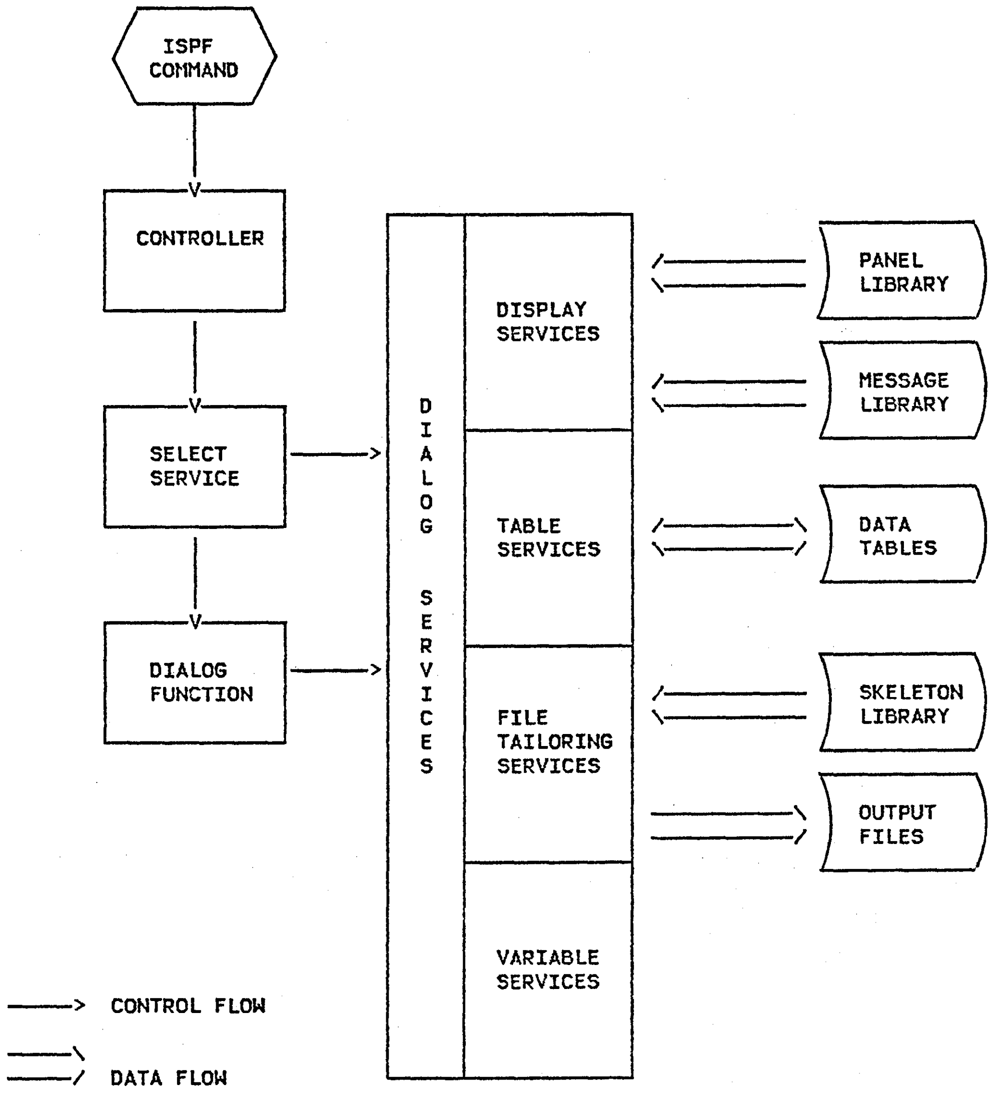


Figure 4. Flow of Control

DIALOG SERVICES OVERVIEW

Once a dialog function receives control, it may use SPF dialog services to continue the interaction with the end user, and to assist in processing operations. The dialog services allow a function to:

- Display predefined screen images and messages
- Build and maintain tables of user information
- Generate output files for job submission or other processing
- Define and control symbolic variables
- Interface to edit and browse, and log hardcopy output
- Control operational modes.

Dialog services may be used only within the SPF environment. They may not be invoked by command procedures or programs running outside of SPF.

Functions coded in a command language may invoke dialog services by means of the ISPEXEC command. Example:

```
ISPEXEC  DISPLAY  PANEL(XYZ)
```

This command invokes the DISPLAY service to display a panel named XYZ.

Caution:

- Functions coded in the CLIST command language must not include attention exits.

- Functions coded in the EXEC2 command language must include a &PRESUME statement prior to issuing an ISPEXEC command. For more information, see "Invocation of Services" in Chapter 4.

Functions coded in a programming language may invoke dialog services by calling a service interface routine, named ISPLINK. Example:

```
CALL  ISPLINK ('DISPLAY', 'XYZ ');
```

ISPLINK is a small program module which is distributed with SPF. It may be called from programs coded in any language that uses standard OS register conventions for call interfaces, and the standard convention for signaling the end of a variable length parameter list.

Data is communicated between the functions and the services by means of "dialog variables." A dialog variable is a character string that may vary in length from zero to 32K bytes. It is referenced symbolically, by name.

For functions coded in a command language, the CLIST or EXEC2 variables are automatically treated as dialog variables; no special action is required to define them to SPF.

For functions coded in a programming language, the internal program variables that are to be used as dialog variables may be identified to SPF via the VDEFINE service, or the program may access and update dialog variables via the VCOPY and VREPLACE services.

Dialog variable names appear in panel, message, and skeleton definitions to allow communication with the functions. A variable name in a panel definition, for example, corresponds exactly to the name of a dialog variable accessible to a function. The variable may be used to initialize information on the panel (prior to display), and to store input entered by the user.

DISPLAY SERVICES

The display services allow a dialog to display information and interpret responses from the end user. There are two display services:

- DISPLAY - Display panel
- TBDISPL - Display table

The DISPLAY service reads panel definitions from the panel library, initializes variable information in the panel from the corresponding dialog variables, and displays the panel on the screen. A message may optionally be displayed with the panel.

After the user has entered information, the user inputs are stored into corresponding dialog variables, and the DISPLAY service returns to the calling function.

The TBDISPL service combines information from panel definitions with information stored in SPF tables. It displays selected columns from all rows in a table, and allows the user to identify rows for processing (one row at a time).

The user may scroll the information up and down, using program function (PF) keys. The amount of scrolling for each depression of a PF key is specified by a scroll field, displayed on the panel, that may be changed by the user. The user may cause a one-time override of the scroll field by entering a scroll value in the command input field, and then pressing one of the Scroll PF keys. See SPF Program Reference for more information on scrolling.

Panel definitions used by the table display service contain non-scrollable text, including column headings, followed by a "model line" that specifies the format of the scrollable data.

Attribute characters in the model line indicate whether each column is protected or unprotected (user-modifiable). Typically, the left-most column is defined as an unprotected selection field. A code entered in that field is interpreted by the dialog function to determine the particular processing for that row.

Panel Definitions

A panel definition consists of up to four sections, of which only the body is required:

1. Attribute section (optional) - defines the special characters that will be used in the body of the panel definition to represent attribute (start of field) bytes. Default attribute characters are provided, which may be overridden.

2. Body (required) - defines the format of the panel as seen by the user, and defines the name of each variable field on the panel.

3. Initialization section (optional) - specifies the initial processing that is to occur prior to displaying the panel. Typically used to define how variables are to be initialized.

4. Processing section (optional) - specifies processing that is to occur after the panel has been displayed. Typically used to define how variables are to be verified and/or translated.

The panel definition syntax is fully described in Chapter 5 of this document.

Panel definitions are created by editing directly into the panel library; no compile or preprocessing step is required. Each panel definition is a member in the library, and is identified by member name.

A sample panel definition is shown in Figure 5. It consists of a panel body followed by an ")END" control statement. It has no attribute, initialization, or processing sections. It uses the default attribute characters, namely:

% (percent sign)	- text (protected) field, high intensity
+ (plus sign)	- text (protected) field, low intensity
_ (underscore)	- input (unprotected) field, high intensity

For text fields, the information following the attribute character is the text to be displayed. Text fields may contain substitutable variables, consisting of a dialog variable name preceded by an ampersand (&). The name and ampersand are replaced with the value of the variable prior to displaying the panel.

For input fields, a dialog variable name immediately follows the attribute character (with no intervening ampersand). The name is replaced with the value of the variable prior to displaying the panel, and any information entered by the user is stored in the variable after the panel has been displayed.

The panel in this example has ten input fields (TYPECHG, LNAME, etc.), indicated with underscores. It also has a substitutable variable (EMPSER) within a text field (on line 2). The first two lines of the panel and the arrows preceding the input fields are all highlighted, indicated with percent signs. The other text fields are low intensity, indicated with plus signs.

Before the panel is displayed, all variables in the panel body will be automatically initialized from the corresponding dialog variables (TYPECHG, LNAME, etc., and EMPSER). After the panel has been displayed, the input fields will be automatically stored into the corresponding dialog variables.

Figure 6 shows the panel as it will appear when displayed, assuming that the current value of EMPSER is "123456", and that the other variables are initially null.

```
%----------------------------  EMPLOYEE RECORDS  ----------------------------
%EMPLOYEE SERIAL: &EMPSER

+   TYPE OF CHANGE%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)

+   EMPLOYEE NAME:
+     LAST   %===>_LNAME         +
+     FIRST  %===>_FNAME         +
+     INITIAL%===>_I+

+   HOME ADDRESS:
+     LINE 1 %===>_ADDR1                                      +
+     LINE 2 %===>_ADDR2                                      +
+     LINE 3 %===>_ADDR3                                      +
+     LINE 4 %===>_ADDR4                                      +

+   HOME PHONE:
+     AREA CODE    %===>_PHA+
+     LOCAL NUMBER%===>_PHNUM   +

)END
```

Figure 5. Sample Panel Definition

```
 ----------------------------  EMPLOYEE RECORDS  ----------------------------
 EMPLOYEE SERIAL: 123456

    TYPE OF CHANGE ===>            (NEW, UPDATE, OR DELETE)

    EMPLOYEE NAME:
      LAST     ===>
      FIRST    ===>
      INITIAL ===>

    HOME ADDRESS:
      LINE 1  ===>
      LINE 2  ===>
      LINE 3  ===>
      LINE 4  ===>

    HOME PHONE:
      AREA CODE     ===>
      LOCAL NUMBER ===>
```

Figure 6. Sample Panel - When Displayed

Message Definitions

When a display service is invoked, a message may optionally be displayed with the panel, or superimposed on the panel that is currently displayed. Messages may also be written to the SPF log file via the LOG service.

Message definitions are created by editing directly into the message library; no compile or preprocessing step is required. Each member of the library may contain several messages. Messages are referenced by message id.

Each message consists of two lines. The first line contains the message id, and optionally:

- Short message text, enclosed in apostrophes (')
- Corresponding help panel (if the user presses the Help PF key)
- Audible alarm indicator (yes or no).

The second line contains the long message text, enclosed in apostrophes.

If a short message is specified, it will be displayed first in the upper right-hand corner of the screen. If the user presses the Help PF key, the long message will then be displayed on line 3 of the screen. If the user presses the Help key again, tutorial mode will be entered.

If a short message is not specified, the long message will be displayed first (on line 3). If the user then presses the Help key, tutorial mode will be entered.

Variable names, preceded by an ampersand (&), may appear anywhere within the short and long message text.

Figure 7 shows an example of a member in the message library. This member contains all message ids which begin with "EMPX21". The message definition syntax is fully described in Chapter 5.

```
EMPX210  'INVALID TYPE OF CHANGE'    .HELP=PERS033   .ALARM=YES
'TYPE OF CHANGE MUST BE NEW, UPDATE, OR DELETE.'

EMPX213  'ENTER FIRST NAME'          .HELP=PERS034   .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE = NEW OR UPDATE.'

EMPX214  'ENTER LAST NAME'           .HELP=PERS034   .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE = NEW OR UPDATE.'

EMPX215  'ENTER HOME ADDRESS'        .HELP=PERS035   .ALARM=YES
'HOME ADDRESS MUST BE ENTERED FOR TYPE OF CHANGE = NEW OR UPDATE.'

EMPX216  'AREA CODE INVALID'         .ALARM=YES
'AREA CODE &PHA IS NOT DEFINED.  PLEASE CHECK THE PHONE BOOK.'

EMPX217  '&EMPSER ADDED'
'EMPLOYEE &LNAME, &FNAME &I ADDED TO FILE.'

EMPX218  '&EMPSER UPDATED'
'RECORDS FOR &LNAME, &FNAME &I UPDATED.'

EMPX219  '&EMPSER DELETED'
'RECORDS FOR &LNAME, &FNAME &I DELETED.'
```

Figure 7. Sample Member in Message Library

TABLE SERVICES

Table services allow sets of dialog variables to be maintained and accessed. A table is a two-dimensional array of information in which each column corresponds to a dialog variable, and each row contains a set of values for those variables.

An example is shown in Figure 8. In this table, the variables that define the columns are:

EMPSER	- Employee Serial Number
LNAME	- Last Name
FNAME	- First Name
I	- Middle Initial
PHA	- Home Phone: Area Code
PHNUM	- Home Phone: Local Number

EMPSER	LNAME	FNAME	I	PHA	PHNUM
598304	Roberston	Richard	P	301	840-1224
172397	Smith	Susan	A	301	547-8465
813058	Russell	Charles	L	202	338-9557
395733	Adams	John	Q	202	477-1776
502774	Caruso	Vincent	J	914	294-1168

Figure 8. Sample Table

Table Residency

A table may be defined as temporary or permanent. A temporary table is created in virtual storage, and deleted upon completion of processing. A permanent table resides on direct access storage. It may be opened for update or for read-only access, at which time the entire table is read into virtual storage. An ENQ is automatically issued to prevent multiple access to a table which is being updated.

Permanent tables are maintained in one or more table libraries, in which each member contains an entire table. The ENQ that occurs when a table is read into virtual storage applies only to that table (member); not the entire library.

When a permanent table is opened for processing, it is read from a table input library. When the table is saved, it is written to a table output library that may be different from the input library. The input and output libraries should be the same if the updated version of the table is to be reopened for further processing by the same dialog. See discussion of library setup in Chapter 3 for specification of input and output libraries.

Accessing of Data

The variable names that define the columns of a table are specified when the table is created. At the same time, one or more columns (variable names) may be specified as keys for accessing the table. For the table shown in Figure 8, EMPSER might be defined as the key variable. Or EMPSER and LNAME might both be defined as keys, in which case a row would be found only if EMPSER <u>and</u> LNAME both match the current values of those variables.

A table may also be accessed by one or more "argument" variables, that need not be key variables. The variables that constitute the search argument may be defined dynamically by means of the TBSARG and TBSCAN services.

In addition, a table may be accessed by "current row pointer" (CRP). When a table is opened, the CRP is automatically positioned to TOP -- ahead of the first row. The table may be scanned by moving the CRP forward or back. A row is retrieved each time the CRP is moved.

When a row is retrieved from a table, the contents of the row are stored into the corresponding dialog variables. When a row is stored (updated or added), the current contents of the dialog variables are saved in that row.

When a row is stored, a list of "extension" variables may be specified, by name. This allows the variables in that row to be extended beyond what was specified when the table was created. A list of extension variable names for a row may be obtained when the row is read. The list of extension variables must be respecified whenever the row is rewritten. Otherwise the extensions to the row will be deleted.

General Services

The following services operate on an entire table:

TBCREATE	Create a new table and open it for processing.
TBOPEN	Open an existing (permanent) table for processing.
TBQUERY	Obtain information about a table.
TBSAVE	Save a permanent copy of a table without closing.
TBCLOSE	Close a table, and save a permanent copy if the table was opened in WRITE mode.
TBEND	Close a table without saving.
TBERASE	Delete a permanent table from the table library.

Temporary tables are created by TBCREATE (NOWRITE mode) and deleted by either TBEND or TBCLOSE. A new permanent table is created by TBCREATE (WRITE mode). This simply creates the table in virtual storage. It does not become permanent until it is stored on direct access storage by either TBSAVE or TBCLOSE.

An existing permanent table is opened and read into virtual storage by TBOPEN. If the table is to be updated (WRITE mode), the new copy is saved by either TBSAVE or TBCLOSE. If it is not to be updated (NOWRITE mode), the virtual storage copy is deleted by either TBEND or TBCLOSE.

Row Operations

The following services operate on a row of the table:

TBADD	Add a new row to the table.
TBDELETE	Delete a row from the table.
TBGET	Retrieve a row from the table.
TBPUT	Update an existing row in the table.

TBMOD	Update a row in the table if it exists (if the keys match); otherwise, add a new row to the table.
TBEXIST	Test for the existence of a row (by key).
TBSCAN	Search a table for a row that matches a list of "argument" variables, and retrieve the row.
TBSARG	Establish a new search argument for use with TBSCAN.
TBTOP	Set CRP to TOP (ahead of the first row).
TBBOTTOM	Set CRP to BOTTOM and retrieve the last row.
TBSKIP	Move the CRP forward or back by a specified number of rows, and then retrieve a row.
TBVCLEAR	Set dialog variables (that correspond to variables in the table) to null.

Storage Requirements

The length of any row in a table cannot exceed 65,536 bytes. The length can be computed as follows:

Row size = 22 + 4a + b + 9c

where:

a = total number of variables in the row including extensions
b = total length of variable data in the row
c = total number of extension variables in the row

The total table size is the sum of the row lengths, plus the length of the data table control block (DTCB). The length of the DTCB can be computed as follows:

DTCB length = 88 + 16d

where:

d = total number of columns (not including extension variables) in the table

The number of tables that may be processed at one time is limited only by the amount of available virtual storage.

FILE TAILORING SERVICES

File tailoring services read skeleton files from a library and write tailored output that may be used to drive other functions. Frequently, file tailoring is used to generate job files for batch execution.

The file tailoring output may be directed to a library or sequential file that has been specified by the dialog function, or it may be directed to a temporary sequential file provided by SPF. The name of the temporary file is available in system variable ZTEMPF. For MVS, ZTEMPF contains a data set name; for VM it contains a file name.

Each skeleton file is read record by record. Each record is scanned to find any dialog variable names (preceded by an ampersand). When a variable name is found, its current value is substituted.

Skeleton file records may also contain statements that control processing. These provide the ability to:

- Set dialog variables
- Imbed other skeleton files
- Conditionally include records
- Iteratively process records in which variables from each row of a table are substituted.

In the latter case, file tailoring services will read the selected rows from the table. If the table was already open (prior to processing the skeleton), it will remain open with the CRP positioned to TOP. If the table was not already open, file tailoring will open it automatically and close it upon completion of processing.

A sample skeleton file is shown in Figure 9. It contains MVS job control language (JCL) for an assembly and optional load-go. In an MVS environment, the tailored output could be submitted to the background for execution. In a VM environment, it could be punched to an MVS machine for batch execution.

The sample skeleton references several dialog variables (ASMPARMS, ASMIN, MEMBER, etc.). It also illustrates use of select statements ")SEL" and ")ENDSEL" to conditionally include records. The first part of the example has nested selects to include concatenated macro libraries if the library names have been specified by the user (i.e., if variables ASMMAC1 and ASMMAC2 are not equal to the null variable Z).

In the second part of the example, select statements are used to conditionally execute a load-go step. An imbed statement, ")IM", is used to bring in a separate skeleton for the load-go step.

The file tailoring services are:

FTOPEN	Prepare the file tailoring process, and specify whether the temporary file is to be used for output.
FTINCL	Specify the skeleton to be used, and start the tailoring process.
FTCLOSE	End the file tailoring process.
FTERASE	Erase (delete) an output file created by file tailoring.

```
//ASM     EXEC  PGM=IFOX00,REGION=128K,
//              PARM=(&ASMPARMS)
//SYSIN     DD  DSN=&ASMIN(&MEMBER),DISP=SHR
//SYSLIB    DD  DSN=SYS1.MACLIB,DISP=SHR
)SEL  &ASMMAC1 ¬= &Z
//          DD  DSN=&ASMMAC1,DISP=SHR
)SEL  &ASMMAC2 ¬= &Z
//          DD  DSN=&ASMMAC2,DISP=SHR
)ENDSEL
)ENDSEL
//SYSUT1    DD  UNIT=SYSDA,SPACE=(CYL,(5,2))
//SYSUT2    DD  UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSUT3    DD  UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSPRINT DD  SYSOUT=(&ASMPRT)
)CM   IF USER SPECIFIED "GO", WRITE OUTPUT IN TEMP DATA SET
)CM   THEN IMBED "LINK AND GO" SKELETON
)SEL  &GOSTEP = YES
//SYSGO     DD  DSN=&&&&OBJSET,UNIT=SYSDA,SPACE=(CYL,(2,1)),
//              DISP=(MOD,PASS)
)IM   LINKGO
)ENDSEL
)CM   ELSE (NOGO), WRITE OUTPUT TO USER DATA SET
)SEL  &GOSTEP = NO
//SYSGO     DD  DSN=&ASMOUT(&MEMBER),DISP=OLD
)ENDSEL
//*
```

Figure 9. Sample Skeleton File

VARIABLE SERVICES

Variable services allow a function to define and use "dialog variables." A dialog variable is a character string that may vary in length from zero to 32K bytes. It is referenced symbolically, by name. The name may be from one to eight characters in length, composed of alphameric characters (A-Z, 0-9, #, $, or @) of which the first must not be numeric.

Dialog variables serve as the main communication vehicle between dialog functions and SPF services. They may also be used to communicate between a function and another function.

Referencing Variables from Command Procedures

From command procedures (CLISTs and EXECs), the variables are always referenced implicitly. All CLIST and EXEC2 variables are automatically treated as dialog variables; no special action is required to define them to SPF. The variables are created dynamically either by execution of the command procedure or by the SPF services that the command procedure uses. CLIST example:

```
SET &AAA = 1
ISPEXEC DISPLAY PANEL(XYZ)
SET &CCC = &BBB + &AAA
```

Same example in EXEC2 language:

```
&AAA = 1
ISPEXEC DISPLAY PANEL(XYZ)
&CCC = &BBB + &AAA
```

Variable AAA is created by the command procedure, simply by setting it to a value. The DISPLAY service is then invoked to display panel XYZ. If panel XYZ references variable AAA, its value may be displayed or changed by the user, depending on how the panel is defined.

The same panel may allow the user to enter a value for another variable (BBB). If a variable of that name does not already exist, a CLIST or EXEC2 variable will be created automatically. Its value may then be referenced in subsequent statements.

Referencing Variables from Programs

From program functions (compiled modules), dialog variables that are to be referenced by the function may be explicitly defined to SPF. The function calls the VDEFINE service to identify the name, address, format, and length of one or more variables within the program to be used as dialog variables. Example in PL/I language:

```
DECLARE AAA CHAR(8);
CALL ISPLINK ('VDEFINE', '(AAA)', AAA, 'CHAR', 8);
CALL ISPLINK ('DISPLAY', 'XYZ ');
```

Variable AAA is declared as an internal program variable (character string, length 8). The program calls the VDEFINE service to define it as a dialog variable. The program then calls the DISPLAY service to display panel XYZ. If panel XYZ references variable AAA, its value may be displayed or changed by the user, depending on how the panel is defined.

If panel XYZ allows the user to enter a value for another variable (BBB) and the program has not defined a dialog variable of that name, storage for the variable will be allocated automatically. BBB is then considered an implicit dialog variable associated with this function. The program can access BBB only via an SPF service (VCOPY), since the program does not have addressability to the implicit area. But if the program invokes an SPF service, the service will be able to access and/or modify the variable (see "Variable Access from Services").

The VDELETE service performs the opposite function of VDEFINE. It may be called to specify the names of one or more program variables that are no longer to be treated as dialog variables.

Variables may be defined whenever desired. Defining a variable with the same name as existing function variable causes the previous occurrences to be masked out. Deleting addressibility to a variable causes the previous occurrence (if any) to be restored. This allows a subroutine to use different variables with the same name as used in the main routine.

Scope of Variables

The scope of a dialog variable may be limited to an individual function or shared between functions.

When a variable is created, it is associated with the function that is currently in control and may not be referenced by other functions. When the function completes execution, all of its variables (defined and implicit) are automatically deleted.

When a function invokes a lower level function via the SELECT service, the lower level function has its own set of variables (which may have the same names as variables belonging to other functions). Again, the lower level function may not access the variables of the invoking function.

Data may be passed from one function to another via parameters. In addition, there are two mechanisms that allow sharing of variables between functions:

- Shared variable pool
- User profile

The shared variable pool allows communication of variables between functions that belong to the same application. A function may copy one or more of its variables into the shared pool by means of the VPUT service. Another function may then obtain a copy of the variable by means of the VGET service.

The user profile contains variables that are automatically retained across sessions for each user. Again, a function may use the VPUT and VGET services to copy variables to/from the user profile. Variables in the profile are limited to 255 bytes in length.

Note: The user profile contains variables that are used by the SPF program development facility. New dialogs may access and update those variables via VGET and VPUT. Generation of new variables in the profile should be approached with caution, since the names must not conflict with those already used by SPF. See SPF Installation and Customization for a list of variable names in the profile.

Figure 10 shows an example of variable sharing. Function A uses VPUT to copy its variables (defined or implicit) to the shared variable pool and/or user profile, and function B uses VGET to obtain the variables.

In this document, the terms function variables, shared variables, and profile variables are used to distinguish the scope and accessibility of the variables. The term dialog variable includes all three.

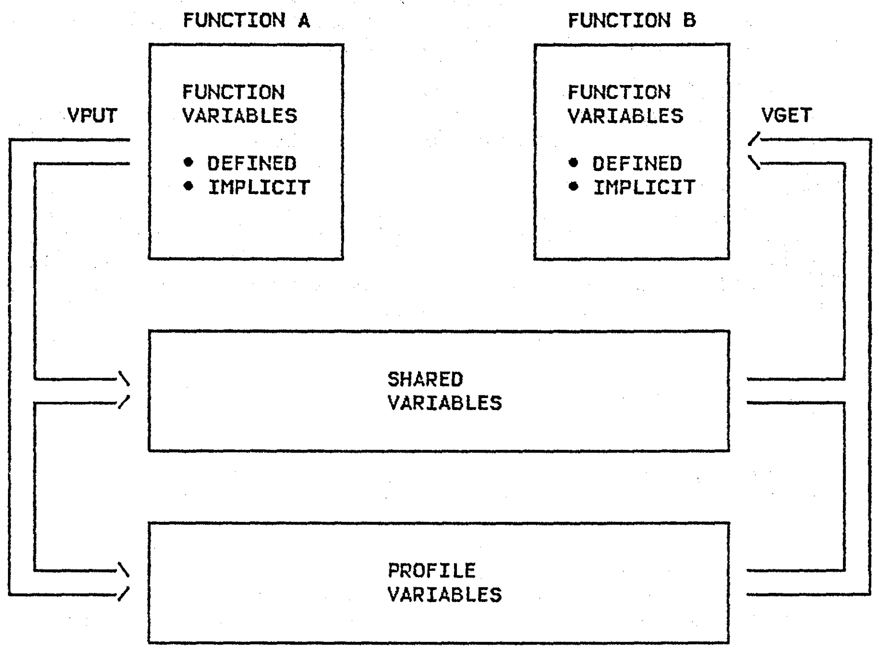


Figure 10. Sharing Variables via VPUT/VGET

Variable Access from Services

When variables are accessed by SPF services, the shared variable pool is logically concatenated to the set of function variables. The search order is:

1. Defined function variables (if any)
2. Implicit function variables (if any)
3. Shared variables

Only the variables for the current function (i.e., the function that invoked the service) are searched. If neither a function variable nor a shared variable of the specified name is found, the current value of the variable is assumed to be null.

Figure 11 shows the logical concatenation of the shared variable pool when a variable is fetched from a service.

When a variable is created or updated by an SPF service, it is stored as a function variable associated with the current function. It is stored in the function's defined area if a variable of that name has been explicitly defined. Otherwise, it is stored in the function's implicit area.

In general, services do not store variables directly into the shared pool nor the user profile. The exceptions are:

- The VPUT service, which is used explicitly for the purpose of copying variables into the shared pool or user profile.
- The SELECT service, when a selection menu (panel) definition references variables in addition to those required by SELECT. See description of selection menus in Chapter 5.

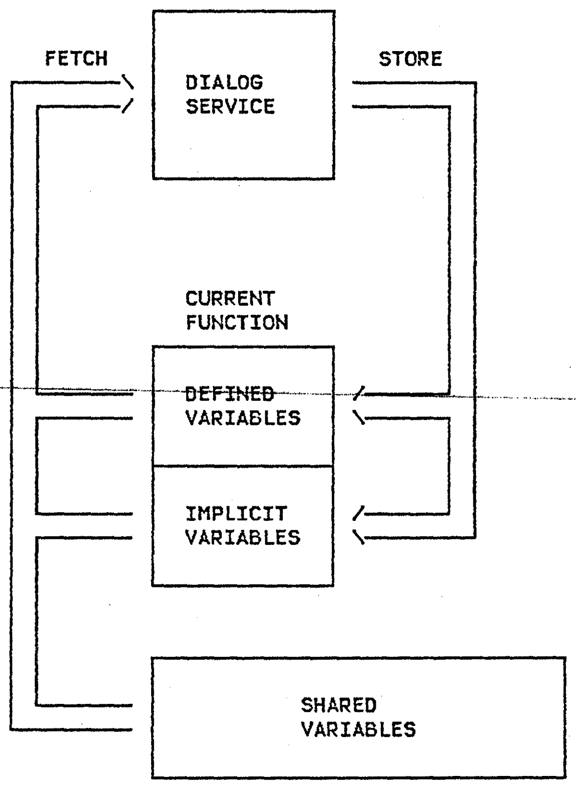


Figure 11. Service Access to Variables

Representation of Variables

Information entered by a user on a panel is in character string format. All dialog variables remain in character string format when stored as implicit function variables, or when stored in the shared pool, the user profile, or SPF tables.

Defined variables, however, may be translated to fixed binary or to a bit string when stored internally in a program module. The internal format is specified when the variable is defined (via VDEFINE). The translation occurs automatically when the variable is stored by an SPF service. A translation back to character string format occurs automatically when the variable is fetched.

When a defined variable is stored, either of two errors may occur:

- Truncation - if the current length of the variable is greater than the defined length within the module.
- Translation - if the variable is defined as other than a character string, and the external representation has invalid characters.

In either case, the SPF service issues a return code of 16.

System Variables

Certain variable names are reserved for use by the system. They all begin with the letter "Z". Dialog developers should avoid names which begin with "Z" when choosing dialog variable names.

Some system variables cannot be modified. They provide the dialog with information about the environment, such as user id, current date and time, and terminal characteristics. These variables reside in the shared variable pool, and may be obtained via the VGET service.

These variables are:

ZUSER	- User id
ZPREFIX	- TSO user prefix[2]
ZLOGON	- Name of TSO LOGON procedure[2]
ZTIME	- Time of day (format hh:mm)
ZDATE	- Current date (format yy/mm/dd)
ZJDATE	- Julian date (format yy.ddd)
ZDAY	- Day of the month (2 characters)
ZMONTH	- Month of the year (2 characters)
ZYEAR	- Year (2 characters)
ZTERM	- Terminal type
ZKEYS	- Number of PF keys
ZTEMPF	- Name of temporary file for file tailoring output
Z	- Null Variable

Other system variables are used for communication of special information between the dialog and the dialog manager. These variables are:

ZERRMSG	- Error message id
ZERRSM	- Short error message text
ZERRLM	- Long error message text
ZERRHM	- Name of help panel associated with error message
ZHTOP	- Top page (panel name) in tutorial
ZHINDEX	- First index page in tutorial
ZTDTOP	- Current top row upon return from table display

The first four are set by SPF services whenever an error condition is encountered (return code of 12 or higher). They are stored in the function variable area.

The variables ZHTOP and ZHINDEX specify the top level table of contents and first index page, respectively, in the tutorial. They should be initialized by the application in the event that the user enters the tutorial via the Help PF key and then requests TOP or INDEX. These two variables should be stored in the shared variable pool.

[2] In the VM environment, ZPREFIX has the same value as ZUSER, and ZLOGON has a null value.

Variable ZTDTOP is set by the table display service (TBDISPL). This variable is stored in the function variable area. See description of TBDISPL for more information.

Summary of Variable Services

The variable services are:

VGET	Retrieve variables from shared pool or profile.
VPUT	Update variables in shared pool or profile.
VDEFINE	Define function variables.
VDELETE	Remove definition of function variables.
~~VCOPY~~	~~Create copy of variable.~~
VREPLACE	Replace variable with copy.
VRESET	Reset function variables.

The first two services, VGET and VPUT, may be invoked from any function. The other variable services are for use from program modules only (not applicable to functions coded in a command language).

OTHER SERVICES

The EDIT and BROWSE interface services allow a dialog function to invoke the SPF editor or browse program, which are part of the program development facility. These services require specification of a data set name (MVS) or fileid (VM), and member name if applicable. The entry panel, which is displayed if edit or browse is selected from the primary option menu, is bypassed.

The LOG service allows a dialog function to write a message to the SPF log file. The end user may specify whether the log is to be printed, kept, or deleted when SPF is terminated.

The CONTROL service allows a dialog function to condition SPF to expect certain kinds of display output, or to control the disposition of errors encountered by SPF services.

The display conditions are:

LINE	Expect line output, not generated by the dialog (e.g., generated by execution of a TSO or CMS command). Optionally, the starting line may be specified for the MVS environment. The starting line is ignored for the VM environment, since line output is always displayed at the top of a blank screen.
SM	Transfer to TSO Session Manager mode on the next line output. [3]
REFRESH	Refresh the entire screen on the next display. Typically used before or after invoking some other full-screen application which is not using SPF display services.
NONDISPL	Do not display the next panel (process the panel without actually displaying it, and simulate the ENTER or End key.)

The disposition of errors may be controlled as follows:

CANCEL	Terminate the dialog function on a error (return code 12 or higher from any service). A message is displayed and logged prior to termination.
RETURN	Return control to the dialog function on all errors (with appropriate return code). A message id is stored in system variable ZERRMSG, which may be used by the dialog function to display and/or log a message.

The default disposition is CANCEL. If a dialog function sets the disposition to RETURN, the change affects only the current function. It does not affect lower level functions invoked via the SELECT service, nor a higher level function when the current function completes.

[3] In the VM environment, the SM condition is treated the same as LINE.

CONTROL FACILITIES

The SPF dialog manager provides control facilities to:

- Display a hierarchy of selection menus and invoke the appropriate dialog functions.
- Transfer in and out of the tutorial, and control the sequence of tutorial pages based on user inputs.
- Manage the physical display image in single screen or split screen mode.
- Interpret program function (PF) key usage for system defined functions.

The display of selection menus is handled by the SELECT service, which has already been discussed under "Flow of Control." The remaining control facilities are described in the following sections.

ONLINE TUTORIAL

A tutorial is a set of panels that provide online information to the end user. The program that displays tutorial pages is part of the dialog manager. It may be entered in either of two ways:

- As a selectable option from a menu, or
- Indirectly from any non-tutorial panel when the user presses the Help PF key.

Transfer in and out of the tutorial via the Help key is transparent to the dialog functions.

Tutorial panels are arranged in a hierarchy. When the tutorial is entered from a menu, the first panel to be displayed is normally the top of the hierarchy. When the tutorial is entered via the Help PF key, the first panel to be displayed is some appropriate panel within the hierarchy, depending upon what the user was doing when the Help key was pressed.

When viewing the tutorial, the user may select topics by number (or other appropriate selection code), or simply press the ENTER key to view the next topic. On any panel, the user may also enter the following commands:

```
BACK  or B - to back up to the previously viewed panel
SKIP  or S - to skip to the next topic
UP    or U - to display a higher level list of topics
TOP   or T - to display the table of contents
INDEX or I - to display the tutorial index.
```

When the user has finished viewing the tutorial, the panel from which the tutorial was entered is redisplayed.

SCREEN MANAGEMENT

At any time during a dialog, the end user may partition the display screen into two "logical" screens. The two logical screens are treated as though they were independent terminals. The dialog manager provides control for mapping the two logical screens onto the physical screen.

In split screen mode, one or the other of the logical screens is considered active at any point in time. The location of the cursor is used to identify which of the two screens is active.

Split screen mode is entered by means of the Split PF key, which may also be used to reposition the split line. Split screen mode is terminated by ending the application on either logical screen. The remaining logical screen is then expanded to its full size.

Use of split screen mode and positioning of the split line is under control of the end user, and totally transparent to the dialog function. Panels that are displayed by the DISPLAY service always pertain to a logical screen.

PROGRAM ACCESS AND FUNCTION KEYS

The dialog manager supports display terminals that have two program access (PA) keys, and 12 or 24 program function (PF) keys. Some keys have system-defined meanings; these are handled by the dialog manager, and are transparent to the dialog function except for the End key.

Other keys may be equated to application-defined commands; these are passed through to the dialog function, as if the user had typed the command in the first input field of the panel, and then pressed the ENTER key.

The two PA keys have system-defined meanings. They may not be redefined by the user.

ATTENTION	(PA1)	Under TSO, the PA1 key is ignored when a panel is displayed with the keyboard unlocked. Pressing PA1 a second time, however, causes the function to be cancelled and the primary option menu to be redisplayed. When a dialog function is executing (keyboard locked), pressing RESET followed by PA1 causes the function to be cancelled and the primary option menu to be redisplayed. Under CMS, the PA1 key causes an immediate return to CP mode. This key should not be used, since it bypasses normal SPF termination.
RESHOW	(PA2)	Redisplays the contents of the screen.

The system-defined PF key operations are described below. The default key assignments are shown in parentheses. For 24-key terminals, PF keys 1-12 have the same defaults as keys 13-24.

HELP	(PF1/13)	Displays additional information about a message or causes a transition into the tutorial.
SPLIT	(PF2/14)	Causes split screen mode to be entered, or changes the location of the split line.
END	(PF3/15)	Terminates the current operation and returns to the previous menu. If the primary option menu is displayed, this key terminates the application.
RETURN	(PF4/16)	Causes an immediate return to the primary option menu. (Logically equivalent to repeated use of the End key.) May also be used to jump directly from one function to another, without displaying the primary option menu, as follows: In any menu or panel input field that is preceded by an arrow (===>), enter an equal sign (=) followed by a primary option. Then press the Return PF key rather than ENTER.

FIND	(PF5/17)	Repeats the action of the previous FIND command or the FIND part of the most recent CHANGE command. Applies to browse and edit only; not used unless the dialog invokes the browse or edit service.
CHANGE	(PF6/18)	Repeats the action of the previous CHANGE command. Applies to edit only; not used unless the dialog invokes the edit service.
UP	(PF7/19)	Causes a scroll up. Applies to browse, edit, and table display only.
DOWN	(PF8/20)	Causes a scroll down. Applies to browse, edit, and table display only.
SWAP	(PF9/21)	Moves the cursor to wherever it was previously positioned on the other logical screen.
LEFT	(PF10/22)	Causes a scroll left. Applies to browse and edit only.
RIGHT	(PF11/23)	Causes a scroll right. Applies to browse and edit only.
CURSOR	(PF12/24)	Moves the cursor to the first input field on line 2 (normally, the option selection or command input field). Pressing this PF key again causes the cursor to be moved to the second input field on line 2, if any (normally the scroll field).
PRINT	(none)	Causes a "snapshot" of the screen image to be recorded in the SPF list file.
PRINT-HI	(none)	Same as PRINT except that high intensity characters on the screen are printed with overstrikes to simulate the dual intensity display.
NOP	(none)	Causes the PF key to be functionless.

The scroll keys are used if the dialog function invokes the table display service (TBDISPL) or the interfaces to edit and browse. During execution of the tutorial, the four scroll PF keys are interpreted as follows:

UP	- same as UP command
DOWN	- same as SKIP command
LEFT	- same as BACK command
RIGHT	- same as ENTER key (display next page).

The PRINT, PRINT-HI, and NOP functions have no default PF key assignments.

The end user may rearrange the system-defined keys, and may redefine system keys to application-defined commands. The only system key function that is required is the End key.

The SPF parms option (program development facility, option 0.3) is used to rearrange or redefine PF keys. See SPF Program Reference. The SPF parms option may be invoked from other selection menus, such as the primary option menu of another application. For an example, see discussion of the master application menu in Chapter 5.

CHAPTER 3. SPF INVOCATION

This chapter describes the library setup requirements in the MVS and VM environments, and the ISPF command used to invoke SPF.

LIBRARY SETUP - MVS ENVIRONMENT

Required and optional libraries for the operation of SPF in the MVS environment are described in this section.

REQUIRED LIBRARIES

The following libraries (partitioned data sets) are required for operation of SPF in the MVS/TSO environment:

DDNAME	DESCRIPTION	RECFM	LRECL	BLKSIZE
ISPPLIB	Panel Library	FB	80	3120
ISPMLIB	Message Library	FB	80	3120
ISPSLIB	Skeleton Library	FB	80	3120
ISPPARM	User Profile Library	F	6000	6000

The panel, message, and skeleton libraries are distributed with SPF. The user profile library is dynamically generated and updated during execution of SPF. The recommended data set names for these libraries are shown below. Check with your system programmer to determine if these are the actual data set names for your installation.

DDNAME	DSNAME
ISPPLIB	ISP.R1M0.ISPPLIB
ISPMLIB	ISP.R1M0.ISPMLIB
ISPSLIB	ISP.R1M0.ISPSLIB
ISPPARM	ISP.R1M0.ISPPARM

Application libraries for panels, messages, and skeletons should be concatenated ahead of the corresponding SPF libraries using the ddnames shown above. They must all have a record format of FB, a logical record length of 80, and a block size of 3120 or greater. (The block size must be a multiple of 80.)

Example. Suppose application XYZ uses the following partitioned data sets for panels, messages, and skeletons:

```
XYZ.PANELS
XYZ.MSGS
XYZ.SKELS
```

The following allocations are required:

```
//ISPPLIB   DD DSN=XYZ.PANELS,DISP=SHR
//          DD DSN=ISP.R1M0.ISPPLIB,DISP=SHR

//ISPMLIB   DD DSN=XYZ.MSGS,DISP=SHR
//          DD DSN=ISP.R1M0.ISPMLIB,DISP=SHR

//ISPSLIB   DD DSN=XYZ.SKELS,DISP=SHR
//          DD DSN=ISP.R1M0.ISPSLIB,DISP=SHR

//ISPPARM   DD DSN=ISP.R1M0.ISPPARM,DISP=SHR
```

ISPPARM is reserved for use by SPF. Other data sets should not be concatenated to this ddname.

These allocations must be performed prior to invoking SPF. They may be done in the user's TSO LOGON procedure using DD statements, as shown above, or in a CLIST using corresponding TSO ALLOCATE commands.

TABLE AND FILE TAILORING LIBRARIES

The following data sets are optional, and need be allocated only if an application uses table or file tailoring services.

DDNAME	DESCRIPTION	RECFM	LRECL	BLKSIZE
ISPTLIB	Table Input Library	FB	80	(See note)
ISPTABL	Table Output Library	FB	80	(See note)
ISPFILE	File Tailoring Output	FB	80	(See note)

Note: The block size may be established by the application. It must be a multiple of 80.

The table input and output libraries must both be partitioned data sets. The ddnames that define them may specify the same data set or different data sets. The data sets must be the same if the updated version of a table is to be reprocessed by the same dialog that updated it.

If tables are used, the table input library must be allocated to ddname ISPTLIB (DISP=SHR) prior to invoking SPF. ISPTLIB may specify a concatenated sequence of partitioned data sets.

The table output library must be allocated to ddname ISPTABL prior to use of table services. ISPTABL may be allocated dynamically by the dialog, and freed upon completion of use. It should be allocated with DISP=SHR even though it specifies an output data set; SPF includes ENQ logic to ensure against simultaneous updates. ISPTABL must not specify a concatenated sequence of data sets.

Note: PCF may not be used to protect the table output library from unauthorized updating if the library is allocated DISP=SHR. The library may either be protected via RACF, or allocated with DISP=OLD and protected via PCF.

File tailoring output may be written to a temporary sequential data set provided by SPF. In this case, there is no need for the dialog to allocate a data set. (The temporary data set is allocated automatically.) The fully qualified name of the temporary data set is available in system variable ZTEMPF.

If the temporary data set is not used, file tailoring output may be written to either a partitioned or sequential data set. The data set must be allocated to ddname ISPFILE prior to use of file tailoring services. ISPFILE may be allocated dynamically by the dialog, and freed upon completion. For a sequential data set, ISPFILE must be allocated with DISP=OLD. For a partitioned data set, it may be allocated with DISP=SHR, but may not be protected via PCF unless it is allocated with DISP=OLD. ISPFILE must not specify a concatenated sequence of data sets.

CLIST AND PROGRAM LIBRARIES

Dialog functions that are coded as CLISTs must be in a procedure library that has been allocated to ddname SYSPROC prior to invoking SPF.

Dialog functions that have been coded as programs must be link edited. The load module may reside in a step library, a system link library (such as SYS1.LINKLIB), or the link pack area. Or it may be in the following partitioned data set (RECFM=U):

DDNAME	DESCRIPTION
ISPLLIB	SPF Link Library

This library may be used for testing new dialogs that contain program-coded functions. If used, it must be allocated to ddname ISPLLIB (DISP=SHR) prior to invoking SPF. ISPLLIB may specify a concatenated sequence of partitioned data sets.

ISPLLIB is used as a task library when fetching load modules. It is searched prior to the system link libraries and the link pack area. A step library may not be used if ISPLLIB is allocated.

LIBRARY SETUP - VM ENVIRONMENT

Required and optional libraries for the operation of SPF in the VM environment are described in this section.

Note: Before SPF is invoked, the user's virtual device 191 must be accessed as the A-disk. SPF assumes that this minidisk is available at all times in read/write mode, and that no other user has write access to it.

REQUIRED LIBRARIES

The following libraries (MACLIBs) are required for operation of SPF in the VM/CMS environment:

DDNAME	DESCRIPTION	FILENAME	FILETYPE
ISPPLIB	Panel Library	ISPPLIB	MACLIB
ISPMLIB	Message Library	ISPMLIB	MACLIB
ISPSLIB	Skeleton Library	ISPSLIB	MACLIB

These libraries are distributed with SPF. They must reside on minidisks that are accessable to the SPF user.

Application libraries for panels, messages, and skeletons should be concatenated ahead of the corresponding SPF libraries using FILEDEF statements with the ddnames shown above.

Example. Suppose application XYZ uses the following libraries for panels, messages, and skeletons:

```
XYZPANLS  MACLIB
XYZMSGS   MACLIB
XYZSKELS  MACLIB
```

The following FILEDEFs are required, assuming that the minidisks containing the XYZ libraries and the distributed SPF libraries have already been linked and accessed.

```
FILEDEF ISPPLIB DISK XYZPANLS MACLIB * (PERM CONCAT)
FILEDEF ISPPLIB DISK ISPPLIB  MACLIB * (PERM CONCAT)

FILEDEF ISPMLIB DISK XYZMSGS  MACLIB * (PERM CONCAT)
FILEDEF ISPMLIB DISK ISPMLIB  MACLIB * (PERM CONCAT)

FILEDEF ISPSLIB DISK XYZSKELS MACLIB * (PERM CONCAT)
FILEDEF ISPSLIB DISK ISPSLIB  MACLIB * (PERM CONCAT)
```

Note: A GLOBAL MACLIB command is not required; SPF dynamically issues GLOBAL commands prior to reading these libraries.

These FILEDEFs must be issued prior to invoking SPF. They may be issued in the user's PROFILE EXEC or in an EXEC that initiates the XYZ application. Any EXEC that invokes SPF must be coded in EXEC2 language.

TABLE AND FILE TAILORING LIBRARIES

The following files are optional, and need be defined only if an application uses table or file tailoring services.

DDNAME	DESCRIPTION
ISPTLIB	Table Input Library
ISPTABL	Table Output Library
ISPFILE	File Tailoring Output

The table input and output libraries must both be MACLIBs. The ddnames that define them may specify the same MACLIB or different MACLIBs. The MACLIBs must be the same if the updated version of a table is to be reprocessed by the same dialog that updated it.

If tables are used, the table input library must be allocated (via a FILEDEF) to ddname ISPTLIB prior to invoking SPF. It may consist of a concatenated sequence of libraries, in which case the FILEDEFs must include the CONCAT parameter (see above example). Again, a GLOBAL MACLIB command is not required.

The table output library must be allocated (via a FILEDEF) to ddname ISPTABL prior to use of table services. If the library does not already exist, the FILEDEF must include a "RECFM F" parameter. The ISPTABL ddname may be allocated dynamically by the dialog, and freed (FILEDEF CLEAR) upon completion of use. ISPTABL must _not_ specify a concatenated sequence of libraries.

File tailoring output may be written to a temporary sequential file provided by SPF. In this case, there is no need for the dialog to allocate an output file. The temporary file is written on the user's A-disk. The file name of the temporary file is available in system variable ZTEMPF. The file type is always ISPTEMP.

If the temporary file is not used, file tailoring output may be written to either a MACLIB or sequential file. The MACLIB or sequential file must be allocated (via a FILEDEF) to ddname ISPFILE prior to use of file tailoring services. If the MACLIB or file does not already exist, the FILEDEF must include a "RECFM F" parameter. The ISPFILE ddname may be allocated dynamically by the dialog, and freed (FILEDEF CLEAR) upon completion. ISPFILE must _not_ specify a concatenated sequence of libraries.

Caution: Table output libraries and, in some cases, file tailoring output may need to be on shared minidisks (i.e., minidisks for which multiple users have concurrent write access). SPF ensures the integrity of these minidisks provided all updating is done via SPF services. However, SPF cannot prevent destructive conflicts if other means (e.g., ordinary CMS commands) are used to update shared minidisks. To guard against destructive conflicts, the following procedures are recommended:

- Isolate shared SPF tables and file tailoring output files on minidisks that do not have other types of files.

- Caution users not to update these minidisks except via SPF services.

- Always access these minidisks as read-only extensions of themselves. This will prevent inadvertent updating.

Example:

```
CP LINK XYZ 294 294 MW
ACCESS 294 D/D
FILEDEF ISPTABL DISK XYZTABL MACLIB D (PERM)
```

In this example, the table output library for the application is assumed to be on the XYZ 294 minidisk. The disk is linked in multiwrite (MW) mode to allow concurrent updating by multiple users. However, when the disk is accessed as the D-disk, "D/D" is specified making it a read-only extension of itself. This will prevent inadvertent updating. A FILEDEF for the table output library (ddname ISPTABL) is then issued to specify the particular table library (XYZTABL MACLIB) on the D-disk.

SPF will automatically reaccess the disk, when needed, to write an updated copy of the table. SPF will then restore the original (D/D) access mode.

The same technique should be used when a table library is allocated for both input and output. Example:

```
CP LINK XYZ 294 294 MW
ACCESS 294 D/D
FILEDEF ISPTLIB DISK XYZTABL MACLIB D (PERM)
    .
    .
    .

FILEDEF ISPTABL DISK XYZTABL MACLIB D (PERM)
```

EXEC AND PROGRAM LIBRARIES

Dialog functions that are coded in the EXEC2 language must be in EXEC files on minidisks that have been linked and accessed prior to invoking the EXEC.

Dialog functions that are coded as programs may be invoked in text (object) module format, or they may be link edited and invoked in load module format. They may be in TEXT files on minidisks that have been linked and accessed prior to invoking the function, or they may be members of either of the following two libraries:

DDNAME	DESCRIPTION
ISPXLIB	Text Module Library (TXTLIB)
ISPLLIB	Load Module Library (LOADLIB)

If a TXTLIB is used, it must be allocated (via a FILEDEF) to ddname ISPXLIB. A concatenated sequence of TXTLIBs may be specified, in which case the FILEDEFs must include the CONCAT parameter. A GLOBAL TXTLIB command is not required.

When a text module is invoked (either as a TEXT file or as a member of a TXTLIB), any additional text modules that it calls will be loaded automatically via "automatic call" reference. The called modules must also be TEXT files on an SPF-accessible minidisk or members of the TXTLIB allocated to ddname ISPXLIB.

If a LOADLIB is used, it must be allocated (via a FILEDEF) to ddname ISPLLIB. A concatenated sequence of LOADLIBs may be specified, in which case the FILEDEFs must include the CONCAT parameter. A GLOBAL LOADLIB command is not required.

No automatic call referencing is available with load modules; all load modules must be fully resolved prior to invocation by SPF.

Caution: Load modules may be used only for programs that are reenterable. Nested use of the same load module or concurrent use in split screen will cause the same copy of the load module to be invoked, even if it is marked reenterable.

RESTRICTIONS ON USE OF MODULE FILES

Use of MODULE files, which are non-relocatable, should be avoided whenever possible. Dialog functions that are invoked as programs via the following SELECT keyword:

```
PGM(program-name)
```

must be relocatable (text or load module format). Whenever such a program is loaded into the user area, SPF automatically turns on CMS subset mode to prevent MODULE files from overlaying the relocatable program. SPF turns off subset mode whenever all relocatable programs in the user area have completed operation.

Note: In the split screen environment, subset mode is not turned off until all relocatable programs associated with _both_ logical screens have completed execution. A dialog may control the use of split screen via the CONTROL service.

Dialog functions that are invoked as commands via the following SELECT keyword:

```
CMD(command)
```

may invoke MODULE files if CMS is not currently operating in subset mode. If subset mode is on, any attempt to invoke or load a MODULE file will result in a CMS return code of +1.

RESTRICTIONS ON USE OF GLOBAL COMMANDS

GLOBAL MACLIB commands are dynamically issued by SPF before fetching members from the panel, message, table, and skeleton libraries. GLOBAL TXTLIB and GLOBAL LOADLIB commands are dynamically issued before invoking program-coded dialog functions. As a result, GLOBAL commands issued by a dialog are not preserved across the invocation of most SPF services.

ISPF COMMAND

SPF is invoked using the ISPF command. (An SPF command may be established as an alias of ISPF.) The command may be issued:

- By the user at a terminal,
- From a command procedure (CLIST or EXEC2), or
- During LOGON (from a TSO LOGON procedure or CMS PROFILE EXEC).

When the ISPF command is used to invoke the SPF program development facility, no parameters are required. When it is used to invoke another application, the PANEL, CMD, or PGM keyword must be specified to indicate the first selection menu to be displayed or the first dialog function to receive control. These three keywords are mutually exclusive; only one may be specified.

Notation conventions for command syntax are described in the Preface to this document.

Format for invoking the SPF program development facility:

```
ISPF  [option]
      [TEST/TESTX/TRACE/TRACEX]
```

Format for invoking another application:

```
ISPF  ( PANEL(panel-name)   [OPT(option)]         )
      { CMD(command)                               }
      ( PGM(program-name)   [PARM(parameters)]    )
      [TEST/TESTX/TRACE/TRACEX]
```

option

Specifies an initial option, which must be a valid option on the first selection menu. This causes direct entry to that option without displaying the menu. (The menu is processed in nondisplay mode, as if the end user had entered the option.)

If an initial option is specified as a positional parameter, it pertains to the primary option menu for the SPF program development facility. If an initial option is specified via the OPT keyword parameter, it pertains to the panel specified by the PANEL keyword parameter.

panel-name

Specifies the name of the first selection menu (i.e., the primary option menu) to be displayed.

command

Specifies a command procedure (CLIST or EXEC2) that is to be invoked as the first dialog function. Command parameters may be included within the parentheses. These parameters are passed to the command procedure. A percent sign (%) may precede the CLIST or EXEC2 name to improve performance.

program-name

Specifies the name of a program that is to be invoked as the first dialog function. If the program is coded in PL/I, it must be a MAIN procedure.

In the MVS environment, this parameter must specify the name of a load module that is accessible via the LINK macro.

In the VM environment, this parameter may specify the name of a TEXT file, a member of a TXTLIB, or a member of a LOADLIB. See "Library Setup - VM Environment" for more information.

Note: Dialog developers should avoid the ISP prefix (the SPF component code) in naming dialog functions. Special linkage conventions, intended only for internal SPF use, are used to invoke programs named "ISPxxxxx".

parameters

Specifies input parameters to be passed to the program. The program should not attempt to modify these parameters.

The parameters within the parentheses are passed as a single character string, preceded by a halfword containing the length of the character string, in binary. (The length value does not include itself.) This convention is exactly the same as if the parameters had been passed via a PARM= keyword on a JCL EXEC statement.

Parameters passed from the ISPF command to a PL/I program may be declared on the procedure statement in the standard way:

```
XXX:  PROC (PARM) OPTIONS(MAIN);
      DCL PARM CHAR (nnn) VAR;
```

If the value of the PARM field is to be used as an SPF dialog variable, it must be assigned to a fixed character string because the VDEFINE service cannot handle varying length PL/I strings. The first character of the PARM field must be a slash ('/') since PL/I assumes that any value prior to the slash is a run-time option.

TEST

Specifies that SPF is to be operated in TEST mode.

TESTX

Specifies that SPF is to be operated in extended TEST mode.

TRACE

Specifies that SPF is to be operated in TRACE mode.

TRACEX

Specifies that SPF is to be operated in extended TRACE mode.

See Chapter 6 for a description of the various test and trace modes.

CHAPTER 4. DESCRIPTION OF SERVICES

This chapter contains a description of syntax conventions and return codes for the dialog services, followed by detailed descriptions of each service.

INVOCATION OF SERVICES

Each service description shows the formats for command invocation (used from a CLIST or EXEC2) and call invocation (used from a program module). PL/I syntax is used to show the call formats. Notation conventions are described in the preface of this document.

COMMAND INVOCATION

SPF services are invoked from a command procedure (CLIST or EXEC2) via the ISPEXEC command. Under VM, the following statement must be included in an EXEC2 procedure prior to issuing an ISPEXEC command:

```
    &PRESUME  &SUBCOMMAND  ISPEXEC
```

A subsequent &PRESUME statement with no operands may be used to cancel the subcommand environment for the purpose of issuing other VM commands.

General format for command invocation:

```
    ISPEXEC  service-name  parameter1  parameter2  ...
```

The "service-name" is an alphabetic string of up to eight characters. For some services, the first parameter following the service name is a required positional parameter. Other services do not use a positional parameter.

All other parameters are keyword parameters. They may take either of two forms:

```
    keyword
    keyword(value)
```

Some keyword parameters are required and others are optional, as indicated for each service. Keyword parameters may be coded in any order. If conflicting keywords are coded, the last keyword is used. Any preceding keywords with which it conflicts are ignored.

CLIST or EXEC2 variables, consisting of a name preceded by an ampersand (&), may be used anywhere within the statement as the service name or as a parameter. Each variable is replaced with its current value prior to execution of the ISPEXEC command.

Caution: EXEC2 variables appearing within parentheses must be followed by a blank, preceding the closing parenthesis. Example:

```
    ISPEXEC  DISPLAY  PANEL(&PNAME )
```

Some SPF services allow the names of dialog variables to be passed as parameters. These names should not be preceded with an ampersand unless substitution is desired. Examples:

```
    ISPEXEC  VGET  XYZ
    ISPEXEC  VGET  &VNAME
```

In the first example, XYZ is the name of the dialog variable to be passed. In the second example, variable VNAME contains the name of the dialog variable to be passed.

Some services accept a list of variable names, passed as a single parameter. For example, the syntax for the VGET service is:

```
    ISPEXEC  VGET  name-list  [SHARED/PROFILE]
```

In this case, "name-list" is a positional parameter. It may consist of a list of dialog variable names, enclosed in parentheses and separated by commas or blanks. For command invocation, the parentheses may be omitted if there is only one name in the list. Examples:

```
    ISPEXEC  VGET  (AAA,BBB,CCC)
    ISPEXEC  VGET  (LNAME FNAME I)
    ISPEXEC  VGET  (XYZ)
    ISPEXEC  VGET  XYZ
```

where the last two examples (with or without the parentheses) are equivalent.

In other cases, a list of variable names may be passed as a keyword parameter. For example, the syntax for the TBPUT service is described as:

```
    ISPEXEC  TBPUT  table-name  [SAVE(name-list)]
```

where the parentheses are required by the "keyword(value)" syntax. Again, the names may be separated by commas or blanks. Examples:

```
    ISPEXEC  TBPUT  TBLA  SAVE(LNAME FNAME I)
    ISPEXEC  TBPUT  XTABLE  SAVE(XYZ)
```

CALL INVOCATION

SPF services are invoked from programs by calling the ISPLINK subroutine. Only a single task level is supported. Under MVS, a dialog function may attach a lower level subtask, but the subtask may not invoke SPF services.

General format for call invocation:

```
CALL  ISPLINK (service-name, parameter1, parameter2 ... );
```

The parameters are all positional; they must be coded in the order described for each service. Optional parameters may be omitted in a right-to-left dropout sequence. Also, an optional character string parameter (name or keyword) may be coded as one or more blanks to obtain the default value. This has the same effect as omitting the parameter.

Standard register conventions are used. Registers 2-14 are preserved across the call.

Note: The last parameter in the calling sequence must be indicated with a high-order "1" bit in the last entry of the address list. This high-order bit is automatically generated by PL/I and COBOL call statements. It requires use of the VL keyword in Assembler call statements.

Call statements are shown in PL/I systax. Service names and keyword values are shown as literals, enclosed in apostrophes ('). Example:

```
CALL  ISPLINK ('TBOPEN', table-name, 'NOWRITE');
```

where "table-name" must be supplied either as a literal or as a variable containing the table name.

Some languages, such as COBOL, do not allow literals within a call statement. Use of literals is never required; all parameters may be specified as variables. PL/I example:

```
DECLARE  SERVICE CHAR(8),
         TABLE   CHAR(8),
         OPTION  CHAR(8);
. . .

SERVICE = 'TBOPEN';
TABLE   = 'XTABLE';
OPTION  = 'NOWRITE';
CALL  ISPLINK (SERVICE, TABLE, OPTION);
```

An equivalent example in COBOL would be:

```
WORKING-STORAGE SECTION.
   77  SERVICE     PICTURE A(8).
   77  TABLE       PICTURE A(8).
   77  OPTION      PICTURE A(8).
   . . .

PROCEDURE DIVISION.
   MOVE "TBOPEN" TO SERVICE.
   MOVE "XTABLE " TO TABLE.
   MOVE "NOWRITE" TO OPTION.
   CALL "ISPLINK" USING  SERVICE TABLE OPTION.
```

The following types of parameters may appear in a calling sequence to ISPLINK:

- Service name or keyword. A left-justified character string that must be coded as shown in the description of the particular service. The string may be up to 8 characters long. It need not be delimited by a trailing blank.

- Single name. A left-justified character string. If the string is less than the maximum length for the particular parameter, it _must_ have a trailing blank to delimit the end of the string. The maximum length for most names is 8 characters. The exceptions are data set name, volume serial, and fileid (see description of EDIT and BROWSE services).

- Numeric value. A full word signed binary number.

- Name list - string format. A list of dialog variable names coded as a character string. The string must start with a left parenthesis and end with a right parenthesis. Within the parentheses, the names may be separated with commas or blanks. Example:

  ```
  '(AAA BBB CCC)'
  ```

  Note: For call invocation, the parentheses _must_ be present even if there is only one name in the list.

- Name list - structure format. A list of dialog variable names passed via a structure. The structure must contain the following information in the following order:

  1. Count. Full word binary integer containing the number of names in the list.

  2. Reserved. Full word binary integer that _must_ contain either 0 or 8.

  3. List of names. Each element in the list must be an 8-byte character string. Within each element, the name of the variable must be left-justified with trailing blanks.

RETURN CODES FROM SERVICES

Each service returns a numeric code indicating the results of the operation. For command invocation, the code is returned in the CLIST variable LASTCC, or EXEC2 variable RETCODE. For call invocation, the code is returned in register 15.

Programs coded in PL/I may examine the return code by using the PLIRETV built-in function. The following declare statements are required:

```
DECLARE ISPLINK EXTERNAL ENTRY OPTIONS(ASM RETCODE);
DECLARE PLIRETV BUILTIN;
```

Programs coded in COBOL may examine the return code by using the RETURN-CODE built-in variable.

~~The return codes are grouped into three general categories:~~

- Normal completion (code 0).

- Exception condition (codes 4 and 8). Indicates a condition that is not necessarily an error, but that the dialog should be aware of.

- Error condition (codes 12, 16, and 20). Indicates that the service did not complete, or only partially completed, due to errors.

The action taken in the case of errors (return code 12 or higher) depends upon the current setting of error mode. There are two error modes:

- CANCEL - Display and log a message. Then terminate the dialog and redisplay the primary option menu.

- RETURN - Format an error message (but do not display or log it). Then return to the function that invoked the service, passing back the designated return code.

The dialog may set the error mode via the CONTROL service. The default mode is CANCEL. In CANCEL mode, control is not returned to the function that invoked the service. Hence, the function will never see a return code of 12 or higher, and need not include logic to process these kinds of errors.

In RETURN mode, control will be returned to the function that invoked the service. That function must then have logic to handle return codes of 12 or higher.

The RETURN mode applies only to the function that set it via the CONTROL service. If a lower level function is invoked, it starts out in CANCEL mode. When a function returns to the higher level function that invoked it, the mode in which the higher level function was operating is resumed.

In RETURN mode, an error message is formatted prior to returning to the function. The message id is contained in system variable ZERRMSG. The short and long message text (in which substitutable variables have been resolved) is contained in system variables ZERRSM and ZERRLM, respectively. If a corresponding help panel was specified in the message definition, the name of the help panel is contained in system variable ZERRHM.

The function may display and/or log the message, if desired, simply by invoking the appropriate service with the message id contained in ZERRMSG. Examples:

```
ISPEXEC  DISPLAY  MSG(&ZERRMSG)
ISPEXEC  LOG  MSG(&ZERRMSG)
```

DISPLAY - DISPLAY PANELS AND MESSAGES

The DISPLAY service reads a panel definition from the panel library, initializes variable panel fields from the corresponding dialog variables, and displays the panel on the screen. A message may optionally be displayed with the panel.

After the end user presses ENTER or the End or Return PF key, user inputs are stored into the corresponding dialog variables, and the DISPLAY service returns to the calling function.

```
ISPEXEC  DISPLAY  [PANEL(panel-name)]
                  [MSG(msg-id)]
                  [CURSOR(field-name)]
```

```
CALL  ISPLINK ('DISPLAY' [,panel-name]
                         [,msg-id]
                         [,field-name] );
```

panel-name

Specifies the name of the panel to be displayed.

msg-id

Specifies the identification of a message to be displayed on the panel.

field-name

Specifies the name of the field where the cursor is to be positioned.

All of the parameters are optional. The processing of the panel-name and msg-id parameters is as follows:

- If panel-name is specified and msg-id is not specified, the panel will be read from the panel library, initialized, and displayed without a message.
- If panel-name and msg-id are both specified, the panel will be read from the panel library, initialized, and displayed with the specified message.
- If panel-name is not specified and msg-id is specified, the current panel will be overlayed with a message, without any initialization being performed on the panel.
- If neither panel-name nor msg-id is specified, the current panel will be redisplayed, without a message and without any initialization.

The field-name parameter may be used to control the initial position of the cursor when the panel is displayed. However, the field-name parameter may be overridden by initialization statements in the panel definition. For more information on use of the field-name parameter, see "Default Cursor Positioning" and "Processing Considerations" in Chapter 5.

The following return codes are possible:

0 - Normal completion (ENTER key pressed, no user errors detected).

8 - End or Return PF key pressed.

12 - The specified panel, message, or cursor field could not be found.

16 - Truncation or translation error in storing defined variables.

20 - Severe error.

TBDISPL - DISPLAY TABLE INFORMATION

The TBDISPL service combines information from a panel definition with information stored in an SPF table. It displays all rows from the table, allowing the user to scroll the information up and down, and select rows for processing.

The format of the display is specified via a panel definition, which TBDISPL reads from the panel library. The panel definition contains two input fields (command input and scroll field) and the non-scrollable text, including column headings. It also contains a "model" line that specifies the format of the scrollable data, and additional information that specifies which columns from the table are to be displayed. See the description of panel formats for table display in Chapter 5.

Each line of scrollable data may have one or more input (unprotected) fields, as well as output (protected) fields. The user may modify the input fields, one line at a time, and may also enter commands in the primary input field.

Before TBDISPL is invoked, the table to be displayed must be open, and the CRP positioned to the row in the table that is to correspond to the first line of the scrollable section of the display. (CRP at TOP is valid; it is treated the same as if the CRP were pointing to the first row.)

```
ISPEXEC  TBDISPL  table-name  PANEL(panel-name)
                              [MSG(msg-id)]
```

```
CALL  ISPLINK ('TBDISPL', table-name, panel-name
                         [,msg-id] );
```

table-name

Specifies the name of the table from which the scrollable data will be obtained.

panel-name

Specifies the name of the panel to be displayed.

msg-id

Specifies the identification of a message to be displayed on the panel.

TBDISPL allows the user to scroll the data up and down, and enter primary commands and/or information in a line of scrollable data. Once inputs have been entered, the TBDISPL service performs the following functions:

1. The contents of the primary command field are stored into the corresponding dialog variable.
2. If the user entered information into a line of scrollable data and pressed ENTER or a scroll key, the CRP is positioned to the corresponding row in the table and the row is retrieved (all variables from that row are stored into the corresponding dialog variables).

3. The information entered by the user on that line is then stored into the corresponding dialog variables. This includes all input fields in the line, which may or may not correspond to variables in the table.

4. The row number that corresponds to the first line currently displayed on the screen is stored into the system variable ZTDTOP. (If desired, the dialog function may reposition the CRP to that row before reinvoking TBDISPL, to cause the scrollable data to be positioned as the user last saw it.)

5. TBDISPL then returns to the dialog function.

TBDISPL does not modify information in the table. The dialog function may use the information entered by the user to determine what processing is to be performed, and may modify the table accordingly.

The following return codes are possible:

0 - Normal completion. The user has entered a command and/or modified one line of the scrollable data, and has pressed either ENTER or a scroll key. The CRP is set to the line that was modified. If no line was modified (only a command was entered), the CRP is set to TOP.

4 - The user has attempted to modify more than one line (and may also have entered a command). The first modified line is processed in the normal fashion (variables stored, CRP set to that line). Changes made to the other lines are ignored. The TBDISPL service will not display an error message in this case, but the dialog function may do so if desired.

8 - The user has pressed the End or Return PF key. If a command was entered, it is stored. But if one or more lines were modified, those variables are not stored. The position of the CRP is unpredictable.

12 - The specified panel or message could not be found or the table was not open.

20 - Severe error.

TBCREATE - CREATE A NEW TABLE

The TBCREATE service creates a new table in virtual storage, and opens it for processing.

TBCREATE allows specification of the variable names that correspond to columns in the table. These variables will be stored in each row of the table. Additional "extension" variables may be specified for a particular row when the row is written.

One or more variables may be defined as keys for accessing the table. If no keys are defined, only the current row pointer can be used for update operations.

The WRITE keyword (which is the default) indicates that the table is permanent, to be stored on disk. The disk copy is not actually created until the TBSAVE or TBCLOSE service is invoked.

The NOWRITE keyword indicates that the table is temporary. When processing of a temporary table is complete, it should be deleted via the TBEND or TBCLOSE service.

```
ISPEXEC  TBCREATE  table-name  [KEYS(key-name-list)]

                               [NAMES(name-list)]

                               [WRITE/NOWRITE]

                               [REPLACE]
```

```
CALL  ISPLINK ('TBCREATE', table-name [,key-name-list]

                                      [,name-list]

                                      [,'WRITE/NOWRITE']

                                      [,'REPLACE'] );
```

table-name

Specifies the name of the table to be created. The name may be from one to eight alphameric characters in length, and must begin with an alphabetic character.

key-name-list

Specifies the variables, by name, that are to be used as keys for accessing the table. See section entitled "Invocation of Services" for specification of name lists. If this parameter is omitted, the table will not be accessible by keys.

name-list

Specifies the other (non-key) variables, by name, to be stored in each row of the table. If this parameter is omitted, each row can only contain extension variables that must be specified when the row is written.

WRITE

Specifies that the table is permanent, to be written to disk via the the TBSAVE or TBCLOSE service.

NOWRITE

Specifies that the table is for temporary usage only.

REPLACE

Specifies that an existing table is to be replaced. If a table of the same name is currently open, it is deleted from virtual storage before the new table is created and return code 4 is issued. If the WRITE parameter is also specified and a duplicate table name exists in the table input library, the table is created and return code 4 is issued. The duplicate table is not deleted from the input library.

The following return codes are possible:

0 - Normal completion.

4 - Normal completion -- duplicate table exists but REPLACE was specified.

8 - Table already exists; REPLACE not specified.

12 - Table in use; ENQ failed.

16 - Table input library not allocated with WRITE specified.

20 - Severe error.

TBOPEN - OPEN A TABLE

The TBOPEN reads a permanent table from the table input library into virtual storage, and opens it for processing. TBOPEN should not be issued for temporary tables.

The table input library, specified by ddname ISPTLIB, must be preallocated prior to invoking SPF. ISPTLIB may specify a concatenation of libraries. See library setup requirements in Chapter 3.

An ENQ is issued to ensure that no other user is currently accessing the table. For the WRITE option, it is an exclusive ENQ which remains in affect until the table is closed. For the NOWRITE option, it is a shared ENQ which remains in affect only during the time that the table is read into storage.

Note: The ENQ applies only to the specified table (member) in the table input library -- not to the entire library.

```
ISPEXEC  TBOPEN  table-name  [WRITE/NOWRITE]
```

```
CALL  ISPLINK ('TBOPEN', table-name [,'WRITE'/'NOWRITE'] );
```

table-name

Specifies the name of the table to be opened.

WRITE

Specifies that the table is being accessed for update. The updated table may subsequently be saved on disk via the TBSAVE or TBCLOSE service.

NOWRITE

Specifies read-only access. Upon completion of processing, the virtual storage copy should be deleted via the TBEND or TBCLOSE service.

The following return codes are possible:

0 - Normal completion.

8 - Table does not exist.

12 - Table in use; ENQ failed.

16 - Table input library not allocated.

20 - Severe error.

TBQUERY - OBTAIN TABLE INFORMATION

The TBQUERY service returns information about a specified table, which must be open prior to invoking this service. The number of key fields and their names, as well as the number of all other columns and their names may be obtained. The number of rows and the current row position may also be obtained.

All of the parameters except for table-name are optional. If they are all omitted, TBQUERY simply validates the existence of an open table. (Return code 12 is issued if the table is not open.)

```
ISPEXEC  TBQUERY  table-name  [KEYS(key-name)]
                              [NAMES(var-name)]
                              [ROWNUM(rownum-name)]
                              [KEYNUM(keynum-name)]
                              [NAMENUM(namenum-name)]
                              [POSITION(crp-name)]
```

```
CALL  ISPLINK ('TBQUERY', table-name [,key-name]
                                     [,var-name]
                                     [,rownum-name]
                                     [,keynum-name]
                                     [,namenum-name]
                                     [,crp-name] );
```

table-name

Specifies the name of the table for which information is desired.

key-name

Specifies the name of a variable into which will be stored a list of key variable names contained in the table. The list will be enclosed in parentheses, and the names within the list will be separated by a blank.

var-name

Specifies the name of a variable into which will be stored a list of non-key variable names contained in the table. The list will be enclosed in parentheses, and the names within the list will be separated by a blank.

rownum-name

Specifies the name of a variable into which will be stored the number of rows contained in the table.

keynum-name

Specifies the name of a variable into which will be stored the number of key variables contained in the table.

namenum-name

Specifies the name of a variable into which will be stored the number of non-key variables contained in the table.

crp-name

Specifies the name of a variable into which will be stored the current row pointer (CRP) number for the table. If the CRP is positioned to TOP, the relative row number returned is zero.

Note: The parameters rownum-name, keynum-name, namenum-name, and crp-name all specify the names of variables into which numeric values will be stored. If these are defined variables (in a program module), they may be either full word fixed variables or character string variables.

The following return codes are possible:

0 - Normal completion.

12 - Table is not open.

16 - Not all keys or names returned because insufficient space was provided.

20 - Severe error.

TBSAVE - SAVE TABLE

The TBSAVE service writes the specified table from virtual storage to the table output library. The table output library must be allocated to a ddname of ISPTABL before invoking this service. The table must be open in WRITE mode.

Optionally, the table can be stored under a different name in the output library.

TBSAVE does not delete the virtual storage copy of the table; the table is still open and available for further processing.

```
ISPEXEC  TBSAVE  table-name  [NEWCOPY/REPLCOPY]
                             [NAME(alt-name)]
                             [PAD(percentage)]
```

```
CALL  ISPLINK ('TBSAVE', table-name, ['NEWCOPY/REPLCOPY']
                                     [,alt-name]
                                     [,percentage] );
```

table-name

Specifies the name of the table to be saved.

NEWCOPY

Specifies that the table is to be written at the end of the output library, regardless of whether an update in place would have been successful. This insures that the original copy of the table is not destroyed before a replacement copy has been written successfully.

REPLCOPY

Specifies that the table is to be rewritten in place in the output library. If the existing member is too small to complete the update in place successfully, or if a member of the same name does not exist in the library, the complete table will be written at the end of the output library.

alt-name

Specifies an alternate name for the table. The table will be stored in the output library with the alternate name. If another table already exists in the output library with that name, it will be replaced. If the table being saved exists in the output library with the original name, that copy will remain unchanged.

percentage

Specifies the percentage of padding space, based on the total size of the table. The padding is added to the total size of the table only when the table is written as a new copy. This parameter does not increase the table size when an update in place is performed.

Padding permits future updating in place, even when the table has expanded in size. Should the table expand beyond the padding space, the table is written at the end of the table output library instead of updated in place.

This parameter must have an unsigned integer value. For call invocation, it must be a full word fixed binary integer.

The default value for this parameter is zero.

Note: If both the NEWCOPY and REPLCOPY keywords are omitted, a comparison is made between the virtual storage size of the table and the external size in the table output library. If there is insufficient storage to write the table in-place, it is written at the end of the table output library.

The following return codes are possible:

0 - Normal completion.

12 - Table is not open.

16 - Table output library not allocated.

20 - Severe error.

TBCLOSE - CLOSE AND SAVE TABLE

The TBCLOSE service terminates processing of the specified table and deletes the virtual storage copy, which is no longer available for further processing.

If the table was opened in WRITE mode, TBCLOSE copies the table from virtual storage to the table output library. In this case, the table output library must be allocated to a ddname of ISPTABL before invoking this service. Optionally, the table can be stored under a different name in the output library.

If the table was opened in NOWRITE mode, TBCLOSE simply deletes the virtual storage copy.

```
ISPEXEC  TBCLOSE  table-name  [NEWCOPY/REPLCOPY]
                              [NAME(alt-name)]
                              [PAD(percentage)]
```

```
CALL  ISPLINK ('TBCLOSE', table-name [,'NEWCOPY/REPLCOPY']
                                     [,alt-name]
                                     [,percentage] );
```

table-name

Specifies the name of the table to be closed.

NEWCOPY

Specifies that the table is to be written at the end of the output library, regardless of whether an update in place would have been successful. This insures that the original copy of the table is not destroyed before a replacement copy has been written successfully.

REPLCOPY

Specifies that the table is to be rewritten in place in the output library. If the existing member is too small to complete the update in place successfully, or if a member of the same name does not exist in the library, the complete table will be written at the end of the output library.

alt-name

Specifies an alternate name for the table. The table will be stored in the output library with the alternate name. If another table already exists in the output library with that name, it will be replaced. If the table being saved exists in the output library with the original name, that copy will remain unchanged.

percentage

Specifies the percentage of padding space, based on the total size of the table. The padding is added to the total size of the table only when the table is written as a new copy. This parameter does not increase the table size when an update in place is performed.

Padding permits future updating in place, even when the table has expanded in size. Should the table expand beyond the padding space, the table is written at the end of the table output library instead of updated in place.

This parameter must have an unsigned integer value. For call invocation, it must be a full word fixed binary integer.

The default value for this parameter is zero.

Note: If both the NEWCOPY and REPLCOPY keywords are omitted, a comparison is made between the virtual storage size of the table and the external size in the table output library. If there is insufficient storage to write the table in-place, it is written at the end of the table output library.

The following return codes are possible:

0 - Normal completion.

12 - Table is not open.

16 - Table output library not allocated.

20 - Severe error.

TBEND - CLOSE TABLE WITHOUT SAVING

The TBEND service deletes the virtual storage copy of the specified table, making it unavailable for further processing. The permanent copy (if any) is not changed.

```
ISPEXEC  TBEND  table-name
```

```
CALL  ISPLINK ('TBEND', table-name);
```

table-name

Specifies the name of the table to be ended.

The following return codes are possible:

0 - Normal completion.

12 - Table is not open.

20 - Severe error.

TBERASE - ERASE A TABLE

The TBERASE service deletes a table from the table output library. The table output library must be allocated to a ddname of ISPTABL before invoking this service.

The table must <u>not</u> be open in WRITE mode when this service is invoked.

```
ISPEXEC  TBERASE  table-name
```

```
CALL  ISPLINK ('TBERASE', table-name);
```

table-name

Specifies the name of the table to be erased.

The following return codes are possible:

0 - Normal completion.

8 - Table does not exist in the output library.

12 - Table in use; ENQ failed.

16 - Table output library not allocated.

20 - Severe error.

TBADD - ADD ROW TO TABLE

The TBADD service adds a new row of variables to a table.

For tables with keys, the table is searched to ensure that the new row has a unique key. The current contents of the key variables (dialog variables that correspond to keys in the table) are used as the search argument.

For tables without keys, no duplicate checking is performed.

Regardless of whether the table has keys, the new row is added immediately following the current row, pointed to by the CRP. The CRP is then set to point to the newly inserted row.

The current contents of all dialog variables that correspond to columns in the table, including key variables, are saved in the row.

Additional non-key variables may also be saved in the row. These "extension" variables apply only to this row; not the entire table. The next time the row is updated, the extension variables must be respecified if they are to be rewritten.

```
ISPEXEC  TBADD  table-name  [SAVE(name-list)]
```

```
CALL  ISPLINK ('TBADD', table-name [,name-list] );
```

table-name

Specifies the name of the table to be updated.

name-list

Specifies a list of extension variables, by name, that are to be saved in the row, in addition to the variables specified when the table was created. See section entitled "Invocation of Services" for specification of name lists.

The following return codes are possible:

0 - Normal completion.

8 - Tables with keys: A row with the same key already exists; CRP set to TOP.

12 - Table is not open.

20 - Severe error.

TBDELETE - DELETE ROW FROM TABLE

The TBDELETE service deletes a row from a table.

For tables with keys, the table is searched for the row to be deleted. The current contents of the key variables (dialog variables that correspond to keys in the table) are used as the search argument.

For tables without keys, the row pointed to by the CRP is deleted.

The current row pointer is always updated to point to the row prior to the one that was deleted.

```
ISPEXEC  TBDELETE  table-name
```

```
CALL  ISPLINK ('TBDELETE', table-name);
```

table-name

Specifies the name of the table to be updated.

The following return codes are possible:

0 - Normal completion.

8 - Tables with keys: Row specified by the value in key variables does not exist; CRP set to TOP.
Non-keyed tables: CRP was at TOP and remains at TOP.

12 - Table is not open.

20 - Severe error.

TBGET - RETRIEVE ROW FROM TABLE

The TBGET service fetches a row from the table.

For tables with keys, the table is searched for the row to be fetched. The current contents of the key variables (dialog variables that correspond to keys in the table) are used as the search argument.

For tables without keys, the row pointed to by the CRP is fetched.

The CRP is always set to point to the row that was fetched.

All variables in the row, including keys and extension variables (if any), are stored into the corresponding dialog variables. A list of extension variable names may also be retrieved.

```
ISPEXEC  TBGET  table-name  [SAVENAME(var-name)]
```

```
CALL  ISPLINK ('TBGET', table-name [,var-name] );
```

table-name

Specifies the name of the table to be read.

var-name

Specifies the name of a variable into which will be stored a list of extension variable names contained in the row. The list will be enclosed in parentheses, and the names within the list will be separated by a blank.

The following return codes are possible:

0 - Normal completion.

8 - Tables with keys: Row specified by the value in key variables does not exist; CRP set to TOP.
Non-keyed tables: CRP was at TOP and remains at TOP.

12 - Table is not open.

16 - Variable value has been truncated or insufficient space provided to return all extension variable names.

20 - Severe error.

TBPUT - UPDATE ROW IN TABLE

The TBPUT service conditionally updates the current row of a table.

For tables with keys, the current contents of the key variables (dialog variables that correspond to keys in the table) must match the key of the current row, pointed to by the CRP. Otherwise, the update is not performed.

For tables without keys, the row pointed to by the CRP is always updated.

If the update was successful, the CRP remains unchanged. It continues to point to the row that was updated. The current contents of all dialog variables that correspond to columns in the table, including key variables, are saved in the row.

Additional non-key variables may also be saved in the row. These "extension" variables apply only to this row; not the entire table. The next time the row is updated, the extension variables must be respecified if they are to be rewritten.

```
ISPEXEC  TBPUT  table-name  [SAVE(name-list)]
```

```
CALL  ISPLINK ('TBPUT', table-name [,name-list] );
```

table-name

Specifies the name of the table to be updated.

name-list

Specifies a list of extension variables, by name, that are to be saved in the row, in addition to the variables specified when the table was created. See section entitled "Invocation of Services" for specification of name lists.

The following return codes are possible:

0 - Normal completion.

8 - Tables with keys: The key does not match that of the current row; CRP set to TOP.
Non-keyed tables: CRP was at TOP and remains at TOP.

12 - Table is not open.

20 - Severe error.

TBMOD - MODIFY ROW IN TABLE

The TBMOD service unconditionally updates a row in a table.

For tables with keys, the table is searched for the row to be updated. The current contents of the key variables (dialog variables that correspond to keys in the table) are used as the search argument. If a match is found, the row is updated. If a match is not found, a TBADD is performed, adding the row to the end of the table.

For tables without keys, TBMOD is equivalent to TBADD.

The CRP is always set to point to the row that was updated or added.

The current contents of all dialog variables that correspond to columns in the table, including key variables, are saved in the row.

Additional non-key variables may also be saved in the row. These "extension" variables apply only to this row; not the entire table. The next time the row is updated, the extension variables must be respecified if they are to be rewritten.

```
ISPEXEC  TBMOD  table-name  [SAVE(name-list)]
```

```
CALL  ISPLINK ('TBMOD', table-name [,name-list] );
```

table-name

Specifies the name of the table to be updated.

name-list

Specifies a list of extension variables, by name, that are to be saved in the row, in addition to the variables specified when the table was created. See section entitled "Invocation of Services" for specification of name lists.

The following return codes are possible:

0 - Normal completion. Tables with keys: Existing row updated. Non-keyed tables: New row added to table.

8 - Tables with keys: Keys did not match; new row added to table.

12 - Table is not open.

20 - Severe error.

TBEXIST - DETERMINE IF ROW EXISTS IN TABLE

The TBEXIST service tests for the existence of a specific row in a table with keys.

The current contents of the key variables (dialog variables that correspond to keys in the table) are used to search the table for the row.

This service is not valid for non-keyed tables and causes the CRP to be set to the top.

```
ISPEXEC  TBEXIST  table-name
```

```
CALL  ISPLINK ('TBEXIST', table-name);
```

table-name

Specifies the name of the table to be searched.

The following return codes are possible:

0 - Normal completion; CRP is positioned to specified row.

8 - Tables with keys: Specified row does not exist; CRP set to TOP.
Non-keyed tables: Service not possible; CRP set to TOP.

12 - Table is not open.

20 - Severe error.

TBSARG - DEFINE A SEARCH ARGUMENT

The TBSARG service establishes a search argument for scanning a table via the TBSCAN service.

The search argument is specified via dialog variables that correspond to columns in the table, including key variables. A value of null for one of the dialog variables means that the corresponding table variable is not to be examined during the search.

Extension variables may be included in the search argument by specifying their names via the name-list parameter. The values of these variables will become part of the search argument. A null value in an extension variable is a valid search argument and requires a corresponding null variable in the matching row.

A search argument of the form AAA* means that only the characters up to the "*" are compared.

Note: In a CLIST, the following technique may be used to set a variable to a literal value that ends with an asterisk:

```
SET &X = AAA&STR(*)
```

The position of the CRP is not affected by the TBSARG service.

TBSARG replaces all previously set search arguments for the specified table.

```
ISPEXEC  TBSARG  table-name  [ARGLIST(name-list)]
```

```
CALL  ISPLINK ('TBSARG', table-name [,name-list] );
```

table-name

Specifies the name of the table for which an argument is to be established.

name-list

Specifies a list of extension variables, by name, whose values are to be used as part of the search argument. See section entitled "Invocation of Services" for specification of name lists.

The following return codes are possible:

0 - Normal completion.

8 - All column variables are null and the name-list parameter was not specified; no argument established.

12 - Table is not open.

20 - Severe error.

TBSCAN - SEARCH TABLE

The TBSCAN service searches a table for a row with values that match an argument list. The argument list may be established via the TBSARG service, or specified in the name-list for TBSCAN.

The search is always in a forward direction, starting with the row after the current row, and continuing to the end of the table. If a match is found, the row is retrieved and the CRP is set to that row. All variables in the row, including keys and extension variables (if any), are stored into the corresponding dialog variables. A list of extension variable names may also be retrieved.

Use of the name-list parameter is optional. If specified, it overrides the search argument set by the TBSARG service for this search only. The values of all variables specified in the name-list parameter will become part of the search argument. A value of the form AAA* means that only the characters up to the "*" are compared (see note in TBSARG description). A null value requires a corresponding null value in the matching row.

If the name-list parameter is omitted, a search argument must have been established by a previous TBSARG command. Otherwise, a severe error occurs.

```
ISPEXEC  TBSCAN  table-name  [ARGLIST(name-list)]

                             [SAVENAME(var-name)]
```

```
CALL  ISPLINK ('TBSCAN', table-name [,name-list]

                                    [,var-name] );
```

table-name

Specifies the name of the table to be searched.

name-list

Specifies a list of variables, by name, whose values are to be used as the search argument. See section entitled "Invocation of Services" for specification of name lists.

var-name

Specifies the name of a variable into which will be stored a list of extension variable names contained in the row. The list will be enclosed in parentheses, and the names within the list will be separated by a blank.

The following return codes are possible:

0 - Normal completion.

8 - Row does not exist, no match found; CRP set to TOP.

12 - Table is not open.

16 - Variable value has been truncated or insufficient space provided to return all extension variable names.

20 - Severe error.

TBTOP - SET ROW POINTER TO TOP

The TBTOP service sets the CRP to the top of a table, ahead of the first row.

```
ISPEXEC  TBTOP  table-name
```

```
CALL  ISPLINK ('TBTOP', table-name);
```

table-name

Specifies the name of the table to be used.

The following return codes are possible:

0 - Normal completion.

12 - Table is not open.

20 - Severe error.

TBBOTTOM - SET ROW POINTER TO BOTTOM

The TBBOTTOM service sets the CRP to the last row of a table, and retrieves the row.

All variables in the row, including keys and extension variables (if any), are stored into the corresponding dialog variables. A list of extension variable names may also be retrieved.

```
ISPEXEC  TBBOTTOM  table-name  [SAVENAME(var-name)]
```

```
CALL  ISPLINK ('TBBOTTOM', table-name [,var-name] );
```

table-name

Specifies the name of the table to be used.

var-name

Specifies the name of a variable into which will be stored a list of extension variable names contained in the row. The list will be enclosed in parentheses, and the names within the list will be separated by a blank.

The following return codes are possible:

0 - Normal completion.

8 - Table is empty; CRP set to TOP.

12 - Table is not open.

16 - Variable value has been truncated or insufficient space provided to return all extension variable names.

20 - Severe error.

TBSKIP - MOVE THE ROW POINTER

The TBSKIP service moves the CRP of table forward or backward by a specified number of rows, and then retrieves the row to which it is pointing.

All variables in the row, including keys and extension variables (if any), are stored into the corresponding dialog variables. A list of extension variable names may also be retrieved.

```
ISPEXEC  TBSKIP  table-name  [NUMBER(number)]
                             [SAVENAME(var-name)]
```

```
CALL  ISPLINK ('TBSKIP', table-name [,number]
                                    [,var-name] );
```

table-name

Specifies the name of the table to be used.

number

Specifies the direction and number of rows to move the CRP. This parameter must be a positive or negative integer. A positive integer moves the CRP toward the bottom of the table; a negative integer moves it toward the top. Zero is an allowable value that results in retrieving the current row.

For call invocation, this parameter must be a full word fixed binary number.

If this parameter is omitted, the default value is 1.

var-name

Specifies the name of a variable into which will be stored a list of extension variable names contained in the row. The list will be enclosed in parentheses, and the names within the list will be separated by a blank.

The following return codes are possible:

0 - Normal completion.

8 - CRP would have gone beyond limit of table; CRP set to TOP.

12 - Table is not open.

16 - Variable value has been truncated or insufficient space provided to return all extension variable names.

20 - Severe error.

TBVCLEAR - CLEAR VARIABLES

The TBVCLEAR service sets dialog variables to nulls.

All dialog variables that correspond to columns in the table (specified when the table was created) are cleared. This includes key variables.

The contents of the table are not changed by this service, nor is the position of the CRP.

```
ISPEXEC  TBVCLEAR  table-name
```

```
CALL  ISPLINK ('TBVCLEAR', table-name);
```

table-name

    Specifies the name of the table to be used.

The following return codes are possible:

0 - Normal completion.

12 - Table is not open.

20 - Severe error.

FTOPEN - BEGIN FILE TAILORING

The FTOPEN service begins the file tailoring process. It allows skeleton files to be accessed from the skeleton library, specified by ddname ISPSLIB.

The skeleton library must be preallocated prior to invoking SPF. ISPSLIB may specify a concatenation of libraries. See library setup requirements in Chapter 3.

If output from file tailoring is not to be placed in a temporary file, the desired output file must be allocated to ddname ISPFILE prior to invoking this service. ISPFILE may designate either a ~~library or a sequential file.~~

```
ISPEXEC  FTOPEN  [TEMP]
```

```
CALL  ISPLINK ('FTOPEN' [,'TEMP'] );
```

TEMP

Specifies that the output of the file tailoring process should be placed in a temporary sequential file. The file is automatically allocated by SPF. Its name is available in system variable ZTEMPF.

In the MVS environment, ZTEMPF contains a fully qualified data set name. Generated JCL in this file may be submitted for background execution via the following TSO command:

```
SUBMIT  &ZTEMPF
```

In the VM environment, the temporary file is written to the user's A-disk. The SPF-generated file name is contained in ZTEMPF. The file type is always ISPTEMP. Data in this file may be punched to another virtual machine via the following CMS command:

```
PUNCH  &ZTEMPF ISPTEMP
```

If this parameter is omitted, the output will be placed in the library or sequential file designated by ddname ISPFILE.

The following return codes are possible:

0 - Normal completion.

8 - File tailoring already in progress.

12 - Output file in use; ENQ failed.

16 - Skeleton library and/or output file not allocated.

20 - Severe error.

FTINCL - INCLUDE SKELETON

The FTINCL service specifies the name of the skeleton (member of the skeleton library) that is to be used to produce the file tailoring output.

The optional parameter NOFT indicates that no tailoring is to be performed, i.e., the entire skeleton is to be copied to the output file exactly as-is with no variable substitution nor interpretation of control records.

See Chapter 5 for the skeleton formats.

```
ISPEXEC  FTINCL  skel-name  [NOFT]
```

```
CALL  ISPLINK ('FTINCL', skel-name [,'NOFT'] );
```

skel-name

Specifies the name of the skeleton.

NOFT

Specifies that no tailoring is to be performed on the skeleton.

The following return codes are possible:

0 - Normal completion.

8 - Skeleton does not exist.

12 - Skeleton or table in use; ENQ failed.

16 - Data truncation occurred; or skeleton library and/or output file not allocated.

20 - Severe error.

FTCLOSE - END FILE TAILORING

The FTCLOSE service is used to indicate the final disposition of the file tailoring output, and to terminate the file tailoring process.

A member-name parameter should be specified if the output file is a library. The file tailoring output will be given the specified member name. No error condition results if the member-name parameter is not specified, but the output is not stored in the library.

If the member-name parameter is specified and the output file is sequential, a severe error results.

```
ISPEXEC  FTCLOSE  [NAME(member-name)]
```

```
CALL  ISPLINK ('FTCLOSE' [,member-name] );
```

member-name

Specifies the name of the member in the output library that is to contain the file tailoring output.

The following return codes are possible:

0 - Normal completion.

8 - File not open (FTOPEN was not used prior to FTCLOSE).

12 - Output file in use; ENQ failed.

20 - Severe error.

FTERASE - ERASE FILE TAILORING OUTPUT

The FTERASE service erases (deletes) a member of the file tailoring output library. The library must be allocated to a ddname of ISPFILE prior to invoking this service.

A severe error results if ISPFILE is allocated to a sequential file.

```
ISPEXEC  FTERASE  member-name
```

```
CALL  ISPLINK ('FTERASE', member-name);
```

member-name

Specifies the name of the member that is to be deleted from the output library.

The following return codes are possible:

0 - Normal completion.

8 - Member does not exist.

12 - Output library in use; ENQ failed.

16 - Output library not allocated.

20 - Severe error.

VGET - RETRIEVE VARIABLES FROM POOL OR PROFILE

The VGET service copies values from the shared variable pool or the user profile to the set of function variables. If a function variable of the same name already exists, it will be updated. If not, it will be created.

```
ISPEXEC  VGET  name-list  [SHARED/PROFILE]
```

```
CALL  ISPLINK ('VGET', name-list [,'SHARED'/'PROFILE'] );
```

name-list

Specifies the names of one or more variables to be copied. See section entitled "Invocation of Services" for specification of name lists.

SHARED

Specifies that the variables are to be copied from the shared variable pool.

PROFILE

Specifies that the variables are to be copied from the user profile.

The following return codes are possible:

0 - Normal completion.

16 - Translation error or truncation has occurred during data movement.

20 - Severe error.

VPUT - UPDATE VARIABLES IN POOL OR PROFILE

The VPUT service copies values from the set of function variables to the shared variable pool or to the user profile.

The search order for the variables to be copied is the defined function variables (if any), followed by the implicit function variables (if any), followed by the shared variable pool. If a variable is not found, its value is assumed to be null.

If a variable of the same name already exists in the shared variable pool or the profile, it will be updated. If not, it will be created.

```
ISPEXEC  VPUT  name-list [SHARED/PROFILE]
```

```
CALL  ISPLINK ('VPUT', name-list [,'SHARED'/'PROFILE'] );
```

name-list

Specifies the names of one or more variables to be copied. See section entitled "Invocation of Services" for specification of name lists.

SHARED

Specifies that the variables are to be copied to the shared variable pool.

PROFILE

Specifies that the variables are to be copied to the user profile.

The following return codes are possible:

0 - Normal completion.

16 - Truncation has occurred while copying variables to the user profile.

20 - Severe error.

VDEFINE - DEFINE FUNCTION VARIABLES

The VDEFINE service gives SPF addressiblity to one or more variables within a program module. The format (character string, fixed binary, bit string, or hex) and length must also be defined.

If more than one variable is defined, they must all have the same format and length, and must be located in contiguous storage starting at the specified address. In other words, it must be an array of variables.

```
CALL  ISPLINK ('VDEFINE', name-list, variable,

                    format, length);
```

name-list

Specifies the symbolic names to be used by SPF when referencing these variables.

Note: If only one name is specified, it must still be enclosed in parentheses. See section entitled "Invocation of Services" for specification of name lists.

variable

Specifies the variable (within the program-addressable storage) being defined. This parameter must specify an array of variables if more than one name was specified in the name-list parameter. The number of names in the list determines the dimension of the array.

format

Specifies the format of the data as stored in the module. This is a keyword parameter, which must be one of the following:

CHAR Character string. Within the variable, the data is left-justified and padded on the right with blanks.

No data conversion is performed on fetching and storing a CHAR variable, nor is there any checking for valid characters.

FIXED Fixed binary integer, represented externally by the characters 0-9.

Fixed variables that have a length of 4 bytes (full word) are treated as signed, represented externally by the absence or presence of a leading minus sign (-). They may also have a null value, which is stored as the maximum negative number (X'80000000').

Fixed variables that have a length less than 4 bytes are treated as unsigned. For these variables, a null value is stored as binary zeros, and cannot be distinguished from a zero value.

BIT Bit string, represented externally by the characters 0 or 1. Within the variable, the data is left-justified and padded on the right with binary zeros.

HEX	Bit string, represented externally by the characters 0-9, A-F. Within the variable, the data is left-justified and padded on the right with binary zeros.

The default is CHAR if this parameter is coded as blank.

Note: In PL/I, a character string to be used as a dialog variable must be declared as fixed length, because VDEFINE cannot handle varying length PL/I strings.

length

Specifies the length of the variable storage, in bytes. The maximum length for a FIXED variable is 4 bytes. The maximum length for other types of variables is 32,767 bytes.

Note: This parameter must be a full word binary integer.

The following return codes are possible:

0 - Normal completion.

20 - Severe error.

VDELETE - REMOVE DEFINITION OF FUNCTION VARIABLES

The VDELETE service removes SPF addressibility to previously defined variables within a program module. This service is the opposite of VDEFINE.

```
CALL  ISPLINK ('VDELETE', name-list);
```

name-list

Specifies the symbolic names from which addressability by SPF is to be removed.

<u>Note</u>: If only one name is specified, it must still be enclosed in parentheses. See section entitled "Invocation of Services" for specification of name lists.

The following return codes are possible:

0 - Normal completion.

8 - At least one variable not found.

20 - Severe error.

VCOPY - CREATE COPY OF VARIABLE

The VCOPY service allows a program module to obtain a copy of a dialog variable. The copied data is in character string format, and may be accessed in either "locate" or "move" mode.

In locate mode, the VCOPY service will automatically allocate storage for the data, and return the address and length to the caller. In move mode, the caller first allocates storage for the data, and then invokes VCOPY, passing the address and length of the storage area into which the data is to be copied.

As with other SPF services, the search for the variable starts with the current function's defined area, followed by the function's implicit area, followed by the shared variable pool. If a variable of the specified name is not found, VCOPY issues a return code of 8.

```
CALL  ISPLINK ('VCOPY', var-name, length, variable
                 [,'LOCATE'/'MOVE'] );
```

var-name

Specifies the name of the variable to be copied.

length

Specifies the length of the data. This is a full word binary variable. In move mode, it must be initialized by the caller to specify the size of the storage area into which the data is to be copied. In either locate or move mode, this variable is set by the VCOPY service to the number of bytes of data.

variable

Specifies the variable to receive the data. In locate mode, this must be a full word address variable. It is set by the VCOPY service, and points to the copy of the data. In move mode, this is a character string variable that receives the data.

LOCATE

Specifies that the variable is to be accessed in locate mode.

MOVE

Specifies that the variable is to be accessed in move mode.

The following return codes are possible:

0 - Normal completion.

8 - Variable does not exist.

16 - Truncation has occurred during data movement (move mode only).

20 - Severe error.

VREPLACE - REPLACE VARIABLE

The VREPLACE service allows a program module to update a variable from a copy (previously obtained via VCOPY) or from any internal variable. The data must be in character string format.

The data to be copied is not modified. The variable to be updated is the function's own defined variable (if it exists) or an implicit variable associated with the function. If the named variable does not exist, it will be created as an implicit function variable.

```
CALL  ISPLINK ('VREPLACE', var-name, length, value);
```

var-name

Specifies the name of the variable to be updated.

length

Specifies the curent length of the data. This is a full word binary integer.

value

Specifies the value of the data. This is a character string variable that contains the data.

The following return codes are possible:

0 - Normal completion.

16 - Truncation has occurred during data movement.

20 - Severe error.

VRESET - RESET FUNCTION VARIABLES

The VRESET service allows a program module to reset its function variables.

Any defined variables are removed from addressability by SPF (as if VDELETEs had been done) and any implicit variables are deleted. The function variables are then in the same state as when a function first receives control.

```
CALL  ISPLINK ('VRESET');
```

The following return codes are possible:

0 - Normal completion.

20 - Severe error.

OTHER SERVICES

SELECT - SELECT PANEL OR FUNCTION

The SELECT service may be used to display a hierarchy of selection menus, or invoke a function.

```
ISPEXEC  SELECT  / PANEL(panel-name)   [OPT(option)]        \
                {  CMD(command)                              }
                 \ PGM(program-name)   [PARM(parameters)]   /
                ~~[NEWAPPL/NEWPOOL]~~
```

```
CALL  ISPLINK ('SELECT', buf-length, buffer);
```

panel-name

Specifies the name of a selection menu to be displayed.

option

Specifies an initial option, which must be a valid option on the menu specified by panel-name. This causes direct entry to that option without displaying the menu. (The menu is processed in nondisplay mode, as if the end user had entered the option.)

command

Specifies a command procedure (CLIST or EXEC2), or any TSO or CMS command that is to be invoked as a dialog function. Command parameters may be included within the parentheses. A percent (%) sign may precede the name of a command procedure (CLIST or EXEC2) to improve performance.

Note: Under TSO, ordinary commands (command processors) are invoked via the ATTACH macro and may not issue SPF dialog services.

program-name

Specifies the name of a program that is to be invoked as a dialog function. If the program is coded in PL/I, it must be a MAIN procedure.

In the MVS environment, this parameter must specify the name of a load module that is accessible via the LINK macro.

In the VM environment, this parameter may specify the name of a TEXT file, a member of a TXTLIB, or a member of a LOADLIB. See "Library Setup - VM Environment" for more information.

Note: Dialog developers should avoid the ISP prefix (the SPF component code) in naming dialog functions. Special linkage conventions, intended only for internal SPF use, are used to invoke programs named "ISPxxxxx".

parameters

Specifies input parameters to be passed to the program. The program should not attempt to modify these parameters.

The parameters within the parentheses are passed as a single character string, preceded by a halfword containing the length of the character string, in binary. (The length value does not include itself.) This convention is exactly the same as if the parameters had been passed via a PARM= keyword on a JCL EXEC statement.

Parameters passed from the SELECT service to a PL/I program may be declared on the procedure statement in the standard way:

```
XXX:  PROC (PARM) OPTIONS(MAIN);
      DCL PARM CHAR (nnn) VAR;
```

If the value of the PARM field is to be used as an SPF dialog variable, it must be assigned to a fixed character string, because the VDEFINE service cannot handle varying length PL/I strings. The first character of the PARM field must be a slash ('/'), because PL/I assumes that any value prior to the slash is a run-time option.

NEWAPPL

Specifies that a new application is being invoked. The next selection menu to be displayed is treated as the primary option menu for the new application. Subsequent use of the Return PF key will cause this menu to be redisplayed.

The "next selection menu" is the one specified by the panel-name parameter in this invocation of SELECT. If the panel-name parameter is not specified, the first invocation of SELECT (from the new application) in which a panel-name parameter _is_ specified will be used to determine the primary option menu for the application.

A new shared variable pool is also created for the new application.

NEWPOOL

Specifies that a new shared variable pool is to be created without specifying a new application. Upon return from the SELECT service, the current shared variable pool will be reinstated.

Note: The only difference between NEWPOOL and NEWAPPL concerns the use of the Return PF key. If the next selection menu is to be treated as a primary option menu on which the Return PF key will stop, NEWAPPL should be specified. Otherwise, NEWPOOL should be specified.

buf-length

Specifies the length of a buffer containing the selection keywords. This parameter must be a full word binary integer.

buffer

Specifies a buffer containing the selection keywords. This is a character string parameter. The selection keywords in the buffer are specified exactly as they would be coded for the ISPEXEC command. Example:

```
BUFNAME = 'PANEL(ABC) OPT(9) NEWAPPL';
```

In the above example, it is assumed that BUFNAME is the name of the buffer. The apostrophes are part of the syntax of the PL/I assignment statement. They are not stored in the buffer itself.

If a command or program is invoked via SELECT, the return code from the command or program is passed back to the function that invoked SELECT. The following return codes are possible if a selection menu (panel) is specified:

0 - Normal Completion. End PF key pressed from the selected menu.

4 - Normal Completion. Return PF key pressed from the selected menu or from some lower level panel.

12 - The specified panel could not be found.

~~16 - Truncation error in storing the OPT or SEL variable.~~

20 - Severe Error.

CONTROL - SET PROCESSING MODES

The CONTROL service defines certain processing options for the dialog environment. The processing options control two areas: the display screen and error processing.

```
ISPEXEC  CONTROL  { DISPLAY  { LINE [START(line-number)] } }
                  {          { SM   [START(line-number)] } }
                  {          { REFRESH                   } }
                  {                                        }
                  { NONDISPL [ENTER/END]                   }
                  { ERRORS   [CANCEL/RETURN]               }
                  { SPLIT    { ENABLE  }                   }
                  {          { DISABLE }                   }
```

```
CALL  ISPLINK ('CONTROL', type [,mode]
                            [,line-number] );
```

For call invocation:

type may be 'DISPLAY', 'NONDISPL', 'ERRORS', or 'SPLIT'

mode may be 'LINE', 'SM', or 'REFRESH' for type 'DISPLAY'

'ENTER' or 'END' for type 'NONDISPL'

'CANCEL' or 'RETURN' for type 'ERRORS'

'ENABLE' or 'DISABLE' for type 'SPLIT'

DISPLAY

Specifies that a display mode is to be set. The valid modes are LINE, SM, and REFRESH. LINE and SM are in effect until the next display of an SPF panel. REFRESH occurs _on_ the next display of an SPF panel.

LINE

Specifies that terminal line-mode output is expected. The screen will be completely rewritten on the next SPF full-screen write operation, after the line(s) have been written.

line-number

In the MVS environment, this parameter specifies the line number on the screen where the line-mode output is to begin. (The first line on the screen is line number 1.) The screen is erased from this line position to the bottom. If this parameter is omitted or coded as zero, the value defaults to the end of the body of the currently displayed panel.

The line-number parameter must have an integer value. For call invocation, it must be a full word binary integer.

This parameter is meaningful only when entering line mode. It may be specified with the SM keyword, since SM reverts to LINE if the Session Manager is not installed.

In the VM environment, this parameter is ignored. Line mode output is always displayed starting at the top of a blank screen.

SM

Specifies that the TSO Session Manager should take control of the screen when the next line-mode output is issued. If the Session Manager is not installed, the SM keyword is treated the same as LINE.

REFRESH

Specifies that the entire screen image should be rewritten when the next SPF-generated full-screen write is issued to the terminal.

NONDISPL

Specifies that no display output is to be issued to the terminal when processing the next panel definition. This option is in effect only for the next panel; after that, normal display mode is resumed. The ENTER or END keywords specify the user response that should be simulated for this panel.

ENTER

Specifies that the ENTER key is to be simulated as the user response to the NONDISPL processing for the next panel.

END

Specifies that the End PF key is to be simulated as the user response to the NONDISPL processing for the next panel.

ERRORS

Specifies that an error mode is to be set. The valid modes are CANCEL and RETURN. If the RETURN mode is set, it applies only to the function that set it via this service.

CANCEL

Specifies that the dialog should be terminated on an error (a return code of 12 or higher from any service). A message will be written to the SPF log file and a panel will be displayed to describe the particular error situation.

RETURN

Specifies that control should be returned to the dialog on an error. The system variable ZERRMSG will contain the message id for a message that describes the error. The message will not be written to the SPF log file, nor will an error panel be displayed.

SPLIT

Specifies that the user's ability to enter split screen mode should be enabled or disabled.

Split screen mode is normally enabled. It is disabled only if explicitly requested via the CONTROL service. It remains disabled until explicitly enabled via the CONTROL service.

The ability to disable split screen mode is available only in the VM environment. If the SPLIT parameter is specified in the MVS environment, a severe error (return code 20) will result.

ENABLE

Specifies that the user should be allowed to enter split screen mode. Pertains to the VM environment only.

DISABLE

Specifies that the user's ability to enter split screen mode should be disabled, until explicitly enabled via the CONTROL service. If the user is already in split screen mode, a return code of 8 is issued and split screen remains enabled.

Pertains to the VM environment only.

The following return codes are posssible:

0 - Normal completion.

8 - Split screen mode already in effect (applies only to a SPLIT DISABLE request); split screen remains enabled.

20 - Severe error.

BROWSE - DISPLAY DATA SET OR FILE

The BROWSE service provides an interface to the SPF browse program, bypassing display of the browse entry panel. See _SPF Program Reference_ for a description of browse.

Syntax for use in an MVS environment:

```
ISPEXEC  BROWSE  DATASET(dsname)  [VOLUME(serial)]
                                  [PASSWORD(pswd-value)]
```

```
CALL  ISPLINK ('BROWSE', dsname [,serial]
                               [,pswd-value] );
```

Syntax for use in a VM environment:

```
ISPEXEC  BROWSE  FILE(fileid)  [MEMBER(member-name)]
```

```
CALL  ISPLINK ('BROWSE', fileid [,member-name] );
```

dsname

Specifies the name of the data set, in TSO syntax, to be browsed. A fully qualified data set name may be specified, enclosed in apostrophes. If the apostrophes are omitted, the TSO user prefix will be automatically left-appended to the data set name.

For partitioned data sets, a member name may be specified, enclosed in parentheses. If a member name is not specified, a member selection list will be displayed.

The maximum length of the dsname parameter is 56 characters.

serial

Specifies the volume serial on which the data set resides. If this parameter is omitted or coded as blank, the system catalog will be searched for the data set name.

The maximum length of the serial parameter is 6 characters.

pswd-value

Specifies the password if the data set has OS password protection. (The password is not specified for RACF or PCF protected data sets.)

fileid

Specifies the fileid, in CMS syntax, to be browsed. The fileid consists of a filename, filetype, and (optionally) filemode, separated by one or more blanks. For call invocation of the browse service, the fileid must be enclosed in parentheses. That is, fileid is one calling sequence parameter consisting of a character string that starts with a left parenthesis and ends with a right parenthesis.

The maximum length of the fileid parameter (including the parentheses for call invocation) is 22 characters.

member-name

Specifies the member to be browsed for a MACLIB or TXTLIB (ignored for other file types). If member name is not specified, a member selection list for the MACLIB or TXTLIB will be displayed.

The following return codes are possible:

0 - Normal completion.

20 - Severe error.

EDIT - EDIT DATA SET OR FILE

The EDIT service provides an interface to the SPF editor, bypassing display of the edit entry panel. See SPF Program Reference for a description of the editor.

Syntax for use in an MVS environment:

```
ISPEXEC  EDIT  DATASET(dsname)  [VOLUME(serial)]
                                [PASSWORD(pswd-value)]
```

```
CALL  ISPLINK ('EDIT', dsname [,serial]
                             [,pswd-value] );
```

Syntax for use in a VM environment:

```
ISPEXEC  EDIT  FILE(fileid)  [MEMBER(member-name)]
```

```
CALL  ISPLINK ('EDIT', fileid [,member-name] );
```

dsname

Specifies the name of the data set, in TSO syntax, to be edited. A fully qualified data set name may be specified, enclosed in apostrophes. If the apostrophes are omitted, the TSO user prefix will be automatically left-appended to the data set name.

For partitioned data sets, a member name may be specified, enclosed in parentheses. If a member name is not specified, a member selection list will be displayed.

The maximum length of the dsname parameter is 56 characters.

serial

Specifies the volume serial on which the data set resides. If this parameter is omitted or coded as blank, the system catalog will be searched for the data set name.

The maximum length of the serial parameter is 6 characters.

pswd-value

Specifies the password if the data set has OS password protection. (The password is not specified for RACF or PCF protected data sets.)

fileid

Specifies the fileid, in CMS syntax, to be edited. The fileid consists of a filename, filetype, and (optionally) filemode, separated by one or more blanks. For call invocation of the edit service, the fileid must be enclosed in parentheses. That is, fileid is one calling sequence parameter consisting of a character string that starts with a left parenthesis and ends with a right parenthesis.

The maximum length of the fileid parameter (including the parentheses for call invocation) is 22 characters.

Note: The EDIT service is intended for use with existing files. In the VM environment, if fileid specifies a non-existent file, the user will be able to create a new file. However, the file characteristics (record format and logical record length) may be unpredictable. They will be whatever was saved in the last-used edit profile for the specified file type. If the user has no edit profile for this file type, the characteristics of the new file will be fixed 80.

member-name

Specifies the member to be edited for a MACLIB or TXTLIB (ignored for other file types). If member name is not specified, a member selection list for the MACLIB or TXTLIB will be displayed.

The following return codes are possible:

0 - Normal completion, data was saved.

4 - Normal completion, data was not saved.

20 - Severe error.

LOG - WRITE MESSAGE TO LOG FILE

The LOG service causes a message to be written to the SPF log file.

```
ISPEXEC  LOG  MSG(msg-id)
```

```
CALL  ISPLINK ('LOG', msg-id);
```

msg-id

Specifies the identification of the message that is to be retrieved from the message library and written to the log.

The following return codes are possible:

0 - Normal completion.

20 - Severe error.

CHAPTER 5. PANEL, MESSAGE, AND SKELETON FORMATS

This chapter contains a detailed description of the syntax for defining panels, messages, and file tailoring skeletons. The description of panel formats is divided into three sections. The first describes the general syntax, the second describes formatting guidelines, and the third describes the specific requirements for selection menus, help/tutorial panels, and table display panels.

PANEL DEFINITIONS - GENERAL SYNTAX

SPF panel definitions are stored in a panel library and displayed by means of the DISPLAY service. Each panel definition is referenced by name, which is the same as the member name in the library.

Panel definitions are created or changed by editing directly into the panel library; no compile or preprocessing step is required.

Each panel definition consists of up to five sections:

1. Attribute section (optional) - defines the special characters that will be used in the body of the panel definition to represent attribute (start of field) bytes. Default attribute characters are provided, which may be overriden.

2. Body (required) - defines the format of the panel as seen by the user, and defines the name of each variable field on the panel.

3. Model section (table display panels only) - defines the format of each line of scrollable data. This section is required for table display panels, and invalid for other types of panels.

4. Initialization section (optional) - specifies the initial processing that is to occur prior to displaying the panel. Typically used to define how variables are to be initialized.

5. Processing section (optional) - specifies processing that is to occur after the panel has been displayed. Typically used to define how variables are to be verified and/or translated.

The sections must appear in the order listed above. The sections are separated with the following header statements:

```
)ATTR   - start of attribute section
)BODY   - start of body
)MODEL  - start of model section
)INIT   - start of initialization section
)PROC   - start of processing section
)END    - end of panel definition
```

In this document, the panel body is described first since it is required in all panel definitions. The attribute section is not described until after the discussion of initialization and processing sections. An attribute section is seldom needed; the default attribute characters will suffice in many cases.

The model section is described under "Panel Definitions - Special Requirements," since it applies to table display panels only.

PANEL BODY

This section of the panel definition specifies the format of the panel as the user sees it. It contains up to 43 records, each of which corresponds to a line on the display.

The section begins with the)BODY header statement, which may be omitted if there is no attribute section and no change to the default attribute characters. The panel body ends with any of the following statements:)MODEL,)INIT,)PROC, or)END.

The special characters defined in the attribute section (or the default attribute characters) are used in the panel body to indicate the start of each field, which is also the end of the preceding field.

The default attribute characters are:

%	(percent sign)	- text (protected) field, high intensity
+	(plus sign)	- text (protected) field, low intensity
_	(underscore)	- input (unprotected) field, high intensity

For text (protected) fields, the information following the attribute character is the text to be displayed. Text fields may contain substitutable variables, consisting of a dialog variable name preceded by an ampersand (&). The name and ampersand are replaced with the value of the variable prior to displaying the panel.

For input (unprotected) fields, a dialog variable name immediately follows the attribute character (with no intervening ampersand). The name is replaced with the value of the variable prior to displaying the panel, and any information entered by the user is stored in the variable after the panel has been displayed.

There is another type of protected field, called an output field, for which there is no default attribute character. Output fields allow padding and justification of the variable information. For more information, see TYPE keyword under "Attribute Section."

A sample panel definition is shown in Figure 12. It consists of a panel body followed by an ")END" control statement. It has no attribute, initialization, or processing sections. It uses the default attribute characters.

This is a data entry panel with ten input fields (TYPECHG, LNAME, etc.), indicated with underscores. It also has a substitutable variable (EMPSER) within a text field (on line 2). The first two lines of the panel and the arrows preceding the input fields are all highlighted, indicated with percent signs. The other text fields are low intensity, indicated with plus signs.

Before the panel is displayed, all variables in the panel body will be automatically initialized from the corresponding dialog variables (TYPECHG, LNAME, etc., and EMPSER). After the panel has been displayed, the input fields will be automatically stored into the corresponding dialog variables.

Figure 13 shows the panel as it will appear when displayed, assuming that the current value of EMPSER is "123456", and that the other variables are initially null.

```
%----------------------------  EMPLOYEE RECORDS  -----------------------------
%EMPLOYEE SERIAL: &EMPSER

+   TYPE OF CHANGE%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)

+   EMPLOYEE NAME:
+     LAST   %===>_LNAME         +
+     FIRST  %===>_FNAME         +
+     INITIAL%===>_I+

+   HOME ADDRESS:
+     LINE 1 %===>_ADDR1                                     +
+     LINE 2 %===>_ADDR2                                     +
+     LINE 3 %===>_ADDR3                                     +
+     LINE 4 %===>_ADDR4                                     +

+   HOME PHONE:
+     AREA CODE   %===>_PHA+
+     LOCAL NUMBER%===>_PHNUM   +

)END
```

Figure 12. Sample Panel Definition

```
 ----------------------------  EMPLOYEE RECORDS  -----------------------------
 EMPLOYEE SERIAL: 123456

    TYPE OF CHANGE ===>           (NEW, UPDATE, OR DELETE)

    EMPLOYEE NAME:
      LAST    ===>
      FIRST   ===>
      INITIAL ===>

    HOME ADDRESS:
      LINE 1  ===>
      LINE 2  ===>
      LINE 3  ===>
      LINE 4  ===>

    HOME PHONE:
      AREA CODE    ===>
      LOCAL NUMBER ===>
```

Figure 13. Sample Panel - When Displayed

INITIALIZATION AND PROCESSING SECTIONS

The initialization section specifies the initial processing that is to occur prior to displaying the panel. It begins with the)INIT header statement and ends with either the)PROC or)END header statement.

The processing section specifies any additional processing that is to occur after the panel has been displayed. It begins with the)PROC header statement and ends with the)END statement.

Note: The automatic initialization of all variables in the panel body occurs after the)INIT section has been processed, just prior to display of the panel. The automatic storing of input fields into the corresponding dialog variables occurs immediately following display, prior to processing of the)PROC section.

Statement Formats

The statements that may be used in the initialization and processing sections are the same, although certain types of statements are typically used only in the initialization section and others only in the processing section.

There are three types of statements that may be used in these sections: assignment, IF, and VER (verify). Two built-in functions may also be used: TRUNC (truncate) and TRANS (translate). These functions may appear on the right hand side of an assignment statement.

The following types of data references may appear within these statements:

- Dialog variable - a name preceded by an ampersand (&).
- Control variable - a name preceded by a period (.) -- see section entitled "Control Variables."
- Literal value - a character string not beginning with an ampersand or period. A literal value may be enclosed in apostrophes ('). It must be enclosed in apostrophes if it begins with a single ampersand or a period, or if it contains any of the following special characters:

  Blank < (+ |) ; ¬ - , > : =

  A literal may contain substitutable variables, consisting of a dialog variable name preceded by an ampersand (&). The name and ampersand are replaced with the value of the variable prior to processing the statement. A double ampersand may be used to specify a literal character string starting with (or containing) an ampersand. See section entitled "Syntax Rules and Restrictions".

In the description of statements and built-in functions that follows, a "variable" may be either a dialog variable or control variable. A "value" may be either type of variable or a literal value.

variable = value

This is an assignment statement. Assignment statements may be used in the)INIT section to set the contents of dialog variables prior to the automatic initialization of variables in the panel body. Assignment statements may also be used in the)PROC section, typically to set the contents of dialog variables that do not correspond to fields in the panel body.

Examples:

```
&A     = ' '
&COUNT = 5
&DSN   = '''SYS1.MACLIB'''
&BB    = &C
```

The first example sets variable A to blanks. The second example sets variable COUNT to a literal character string (the number 5). The third example sets variable DSN to a character string that begins and ends with an apostrophe (see "Syntax Rules and Restrictions"). The fourth example sets variable BB to the contents of another variable, C.

TRUNC (variable,value)

This built-in function may occur on the right hand side of an assignment statement to cause truncation. The first parameter inside the parentheses specifies the variable to be truncated. This is followed by a value that may be a numeric quantity indicating the length of the truncated result, or any special character indicating truncation at the first occurrence of that character. Examples:

```
&A = TRUNC (&XYZ,3)
&INTEG = TRUNC (&NUMB,'.')
```

In the first example, the contents of variable XYZ are truncated to a length of three characters and stored in variable A. (Variable XYZ remains unchanged.) In the second example, the contents of variable NUMB are truncated at the first occurrence of a period and stored in variable INTEG. (Variable NUMB remains unchanged.) If NUMB contains "3.2.4", INTEG will contain "3".

TRANS (variable value,value ... [MSG=value])

This built-in function may occur on the right hand side of an assignment statement to cause translation. The first parameter inside the parentheses specifies the variable to be translated. This is followed by paired values. The first value in each pair indicates a possible value of the variable, and the second indicates the translated result. Example:

```
&REPL = TRANS (&MOD Y,YES N,NO)
```

The current value of variable MOD is translated, and the result is stored in variable REPL. (Variable MOD remains unchanged.) The translation is as follows: If the current value of MOD is "Y", it is translated to "YES". If the current value is "N", it is translated to "NO". If the current value is anything else (neither "Y" nor "N"), it is translated to blank.

The anything-else condition may be specified by using an asterisk in the last set of paired values. Examples:

```
    &REPL = TRANS (&MOD  ...  *,'?')
    &REPL = TRANS (&MOD  ...  *,*)
```

In the first example, if the current value of MOD does not match any of the listed values, a question mark will be stored in variable REPL. In the second example, if the current value of MOD does not match any of the listed values, the value of MOD will be stored untranslated into REPL.

Another option for the anything-else condition is to cause a message to be displayed to the user, by specifying MSG=value, where "value" is a message id. Typically, this technique is used in the processing section of the panel description. Example:

```
    &DISP = TRANS (&D 1,SHR 2,NEW 3,MOD MSG=ISPG001)
```

The contents of variable D are translated as follows: "1" is translated to "SHR", "2" is translated to "NEW", and "3" is translated to "MOD". If none of the listed values is encountered, message ISPG001 is displayed. Message ISPG001 may be an error message indicating that the user has entered an invalid option.

For both the TRANS and TRUNC built-in functions, the source and destination variables may be the same. Figure 14 shows an example in which it is assumed that variable TYPEHG was originally set (in the dialog function) to a single character "N", "U", or "D". In the)INIT section, variable TYPCHG is translated to "NEW", "UPDATE", or "DELETE" and stored into itself prior to display of the panel. In the)PROC section, TYPCHG is truncated back to a single character.

Use of this technique allows the end user to change the valid options for TYPCHG by simply overtyping the first character.

Finally, the TRANS and TRUNC built-in functions may be nested. Examples:

```
    &XYZ = TRUNC( TRANS(&A ---),1 )
    &SEL = TRANS( TRUNC(&OPT,'.') --- )
```

In the first example, the current value of variable A is translated, the translated value is then truncated to a length of one, and the result is stored in variable XYZ. In the second example, the contents of variable OPT are truncated at the first period, the truncated value is then translated, and the result is stored in variable SEL.

```
%----------------------------  EMPLOYEE RECORDS  ----------------------------
%EMPLOYEE SERIAL: &EMPSER

+   TYPE OF CHANGE%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)

+   EMPLOYEE NAME:
+     LAST    %===>_LNAME         +
+     FIRST   %===>_FNAME         +
+     INITIAL%===>_I+

+   HOME ADDRESS:
+     LINE 1 %===>_ADDR1                                  +
+     LINE 2 %===>_ADDR2                                  +
+     LINE 3 %===>_ADDR3                                  +
+     LINE 4 %===>_ADDR4                                  +

+   HOME PHONE:
+     AREA CODE    %===>_PHA+
+     LOCAL NUMBER%===>_PHNUM   +

)INIT
  &TYPECHG = TRANS (&TYPECHG  N,NEW  U,UPDATE  D,DELETE)

)PROC
  &TYPECHG = TRUNC (&TYPECHG,1)

)END
```

Figure 14. Sample Panel with TRANS and TRUNC

IF (variable operator value [,value ...])

The IF statement may be used to test the current value of a variable. The parentheses contain a conditional expression, in which the operator may be either equal (=) or not equal (¬=). One or more values may be specified. Examples:

```
IF (&DSN = ' ')
IF (&OPT = 1,2,5)
IF (&A ¬= &B)
IF (&A ¬= AAA,BBB)
```

The first example is true if variable DSN is null or contains blanks. The second is true if variable OPT contains any of the literal values 1, 2, or 5. The third is true if variable A is not equal to the value of variable B. The fourth is true if variable A is not equal to either of the literal values AAA or BBB, which is the same as saying that variable A is not equal to AAA and not equal to BBB.

The IF statement is indentation sensitive. If the conditional expression is true, processing continues with the next statement. Otherwise all following statements are skipped up to the next statement that begins in the same column as the IF or in a column to the left of the IF. Example:

```
IF (&XYZ = ' ')
  &A = &B
  &B = &PQR
  IF (&B = YES)
    &C = NO
&D = &ZZZ
```

In this example, processing skips to statement &D = &ZZZ from either IF statement if the condition is false.

Figure 15 shows a sample panel with an IF statement. The current value of variable PHA is tested for blank. If it is blank, PHA is initialized to the literal value 301.

```
%---------------------------  EMPLOYEE RECORDS  ----------------------------
%EMPLOYEE SERIAL: &EMPSER

+   TYPE OF CHANGE%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)

+   EMPLOYEE NAME:
+     LAST   %===>_LNAME         +
+     FIRST  %===>_FNAME         +
+     INITIAL%===>_I+

+   HOME ADDRESS:
+     LINE 1 %===>_ADDR1                                     +
+     LINE 2 %===>_ADDR2                                     +
+     LINE 3 %===>_ADDR3                                     +
+     LINE 4 %===>_ADDR4                                     +

+   HOME PHONE:
+     AREA CODE    %===>_PHA+
+     LOCAL NUMBER%===>_PHNUM   +

)INIT
  IF (&PHA = ' ')
    &PHA = 301
  &TYPECHG = TRANS (&TYPECHG  N,NEW  U,UPDATE  D,DELETE)

)PROC
  &TYPECHG = TRUNC (&TYPECHG,1)

)END
```

Figure 15. Sample Panel with IF Statement

VER (variable,keyword [,value ...] [MSG=value])

The verify statement, VER, may be used to check that the current value of a variable meets some criteria. Typically, it is used in the processing section to verify the contents of input fields entered by the user.

The first parameter inside the parentheses specifies the variable to be checked. The second parameter is a keyword indicating the type of verification. The number and meaning of the values that follow the keyword are dependent upon the type of verification.

If the variable does not meet the verification criteria, a message is displayed. The message may be specified via the MSG=value parameter, where "value" is a message id. If no message is specified, an SPF-supplied message is displayed, based on the type of verification.

SPF provides several types of verification, described below. In these descriptions, "xxx" is used to represent the variable name. The values that must follow the verification keyword, if any, are also indicated.

- VER (xxx,NONBLANK) - The variable is required (must not be blank).
- VER (xxx,ALPHA) - The variable must contain all alphabetic characters (A-Z, #, $, or @).
- VER (xxx,NUM) - The variable must contain all numeric characters (0-9).
- VER (xxx,HEX) - The variable must contain all hexadecimal characters (0-9, A-F).
- VER (xxx,PICT,string) - The variable must contain characters that match the corresponding type of character in the picture string. The "string" parameter may be composed of the following characters:

  C - any character
  A - any alphabetic character (A-Z, #, $, or @)
  N - any numeric character (0-9)
  9 - any numeric character (same as "N")
  X - any hexadicimal character (0-9, A-F)

  In addition, the string may contain any special character (except #, $, or @), which represents itself. Example:

  ```
  VER (xxx,PICT,'A/NNN')
  ```

  The value must start with an alphabetic character, followed by a slash, followed by three numeric characters.
- VER (xxx,NAME) - The variable must contain a valid name, following the rules of member names (up to eight alphameric characters of which the first must be alphabetic).
- VER (xxx,DSNAME) - The variable must contain a valid data set name (in TSO syntax).
- VER (xxx,RANGE,lower,upper) - The variable must be numeric, and its value must fall (inclusively) within the specified lower and upper limits.
- VER (xxx,LIST,value1,value2, ...) - The variable must contain one of the listed values.

For all tests except NONBLANK, a blank value is acceptable. That is, if the user enters a value (or leaves a non-blank initial value unchanged), it must conform to the specified condition. But if the user leaves an input field blank, the field will pass any verification test except NONBLANK.

Figure 16 shows a sample panel with VER statements to verify that information entered by the user meets the following criteria:

- The truncated value of TYPCHG is "N", "U", or "D".
- The three name variables (LNAME, FNAME, and I) contain all alphabetic characters.
- The area code (PHA) contains all numeric characters.
- The local number (PHNUM) contains three numeric characters, followed by a hyphen, followed by four numeric characters.

For the TYPECHG test, a message id has been specified in the event that the test fails. In all the other cases, an SPF-provided message will be displayed if the variable fails the verification test.

```
%---------------------------- EMPLOYEE RECORDS ------------------------------
%EMPLOYEE SERIAL: &EMPSER

+   TYPE OF CHANGE%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)

+   EMPLOYEE NAME:
+     LAST   %===>_LNAME        +
+     FIRST  %===>_FNAME        +
+     INITIAL%===>_I+

+   HOME ADDRESS:
+     LINE 1 %===>_ADDR1                                 +
+     LINE 2 %===>_ADDR2                                 +
+     LINE 3 %===>_ADDR3                                 +
+     LINE 4 %===>_ADDR4                                 +

+   HOME PHONE:
+     AREA CODE    %===>_PHA+
+     LOCAL NUMBER%===>_PHNUM   +

)INIT
  IF (&PHA = ' ')
    &PHA = 301
  &TYPECHG = TRANS (&TYPECHG N,NEW U,UPDATE D,DELETE)

)PROC
  &TYPECHG = TRUNC (&TYPECHG,1)
  VER (&TYPECHG,LIST,N,U,D,MSG=EMPX210)
  VER (&LNAME,ALPHA)
  VER (&FNAME,ALPHA)
  VER (&I,ALPHA)
  VER (&PHA,NUM)
  VER (&PHNUM,PICT,'NNN-NNNN')

)END
```

Figure 16. Sample Panel with Verification

Control Variables

Control variables are used to control and/or test certain conditions pertaining to the display of a panel.

The control variables are:

.CURSOR May be set in the initialization section to control the initial placement of the cursor. Its value must be a character string that matches a field name in the panel body. Example:

```
.CURSOR = DSN
```

The cursor is placed at the beginning of field DSN.

.HELP May be set in the initialization section to establish a tutorial (explain) panel to be displayed if the user presses the Help PF key. Example:

```
.HELP = ISPTE
```

If the user presses the Help PF key, tutorial page ISPTE will be displayed.

.MSG May be set to a message id, typically in the processing section, to cause a message to be displayed. Example:

```
.MSG = ISPE016
```

This variable is automatically set via the MSG=value keyword on a TRANS or VER statement.

.RESP Indicates which key the user pressed in response to the panel. Automatically set to either ENTER or END after the panel is displayed. It may be tested in the processing section to determine which key was pressed. Example:

```
IF (.RESP = END)
```

This variable may be set in the initialization section to simulate a user response. In this case, the panel is not displayed but is processed as if the user had pressed ENTER or the End PK key without entering any data.

The control variables are automatically reset (set to blank) when the panel display service first receives control. If .MSG and .CURSOR are still blank after processing of the initialization section, they are set to the values passed via the calling sequence (if any) for an initial message or cursor position.

<u>Note</u>: Under certain conditions, processing of the initialization section is bypassed. See "Processing Considerations" for more information.

Once .MSG and .CURSOR have been set non-blank, they will retain their initial values until the panel is displayed (or redisplayed), at which time they are again reset.

Figure 17 shows an example in which both .HELP and .CURSOR have been set in the)INIT section of the panel definition.

```
%----------------------------  EMPLOYEE RECORDS  ----------------------------
%EMPLOYEE SERIAL: &EMPSER

+   TYPE OF CHANGE%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)

+   EMPLOYEE NAME:
+     LAST   %===>_LNAME         +
+     FIRST  %===>_FNAME         +
+     INITIAL%===>_I+

+   HOME ADDRESS:
+     LINE 1 %===>_ADDR1                                    +
+     LINE 2 %===>_ADDR2                                    +
+     LINE 3 %===>_ADDR3                                    +
+     LINE 4 %===>_ADDR4                                    +

+   HOME PHONE:
+     AREA CODE    %===>_PHA+
+     LOCAL NUMBER%===>_PHNUM    +

)INIT
  .HELP = PERS032
  .CURSOR = TYPECHG
  IF (&PHA = ' ')
    &PHA = 301
  &TYPECHG = TRANS (&TYPECHG N,NEW U,UPDATE D,DELETE)

)PROC
  &TYPECHG = TRUNC (&TYPECHG,1)
  VER (&TYPECHG,LIST,N,U,D,MSG=EMPX210)
  VER (&LNAME,ALPHA)
  VER (&FNAME,ALPHA)
  VER (&I,ALPHA)
  VER (&PHA,NUM)
  VER (&PHNUM,PICT,'NNN-NNNN')

)END
```

Figure 17. Sample Panel with Control Variables

Default Cursor Positioning

If the control variable .CURSOR is not explicitly initialized (or if it is set to blank), the initial position of the cursor will be determined as follows:

- The panel body is scanned from top to bottom, and the cursor is placed at the beginning of the first input field that meets the following criteria:
    - It must be the first or only input field on a line, and
    - It must not have an initial value (i.e., the corresponding dialog variable must be null or blank).
- If no fields meet the above criteria, the cursor will be placed on the first input field in the panel body.
- If the panel has no input fields, the cursor will be placed at row 1, column 1.

Whenever a message is displayed due to a verification failure, MSG=value condition in a TRANS, or explicit setting of .MSG, the cursor is automatically positioned at the beginning of the field that was last referenced in any panel definition statement.

Examples:

```
&XYZ = TRANS (&A ... MSG=xxxxx)
&A = TRANS (&XYZ ... MSG=xxxxx)
VER (&XYZ,NONBLANK)  VER (&B,ALPHA)
```

Assume that field XYZ exists in the panel body, but there are no fields corresponding to variables A or B. In all of the above cases, the cursor would be placed on field XYZ if a message is displayed.

ATTRIBUTE SECTION

This section defines the special characters that will be used in the body of the panel definition to represent attribute (start of field) bytes. When the panel is displayed, these characters are replaced with the appropriate hardware attribute bytes, and appear on the screen as blanks.

The attribute section <u>precedes</u> the panel body. It begins with the)ATTR header statement and ends with the)BODY header statement.

<u>Statement Formats</u>

Each statement in the attribute section must begin with a single character. This defines the attribute character for a particular ~~kind of field. The remainder of the statement contains keyword~~ parameters that define the nature of the field. The keywords that may be specified are described below.

As a general rule, special (non-alphameric) characters should be chosen for attribute characters so that they will not conflict with the panel text. An ampersand (&) is illegal as an attribute character.

The keyword parameters that may be specified to the right of the attribute character are described below. They are all optional, except that at least one parameter must be specified. They may be specified in any order.

TYPE(value)

Defines the type of field. The "value" may be:

TEXT	- text (protected) field
INPUT	- input (unprotected) field
OUTPUT	- output (protected) field

If this keyword parameter is omitted, the default is INPUT.

Text fields are displayed exactly as specified in the body of the panel, except that any variable names (preceded by an ampersand) are replaced with the current value of the variable.

For input and output fields, a dialog variable name must immediately follow the attribute character (with no intervening ampersand). No text may be included within the field.

Input fields are initialized prior to display, and may be entered (or overtyped) by the user. Output fields are initialized prior to display, but may not be changed by the user. Note that both input and output fields may have associated caps, justification, and pad attributes. Note also that that there is no default attribute character for output fields.

INTENS(value)

Defines the intensity of field. The "value" may be:

HIGH	- high intensity field
LOW	- low (normal) intensity field
NON	- non-display field (valid only for input fields)

If this keyword parameter is omitted, the default is HIGH.

CAPS(value)

Defines the upper/lower case attribute of the field, and is valid only for input and output fields. The "value" may be:

ON	- translate to upper case
OFF	- no translation

If this keyword parameter is omitted, the default is ON.

For caps on, initial values and values entered by the user are automatically translated to upper case. For caps off, no translation is performed.

JUST(value)

Defines how the contents of the field are to be justified, and is valid only for input and output fields. The "value" may be:

LEFT	- left justification
RIGHT	- right justification
ASIS	- no justification

If this keyword parameter is omitted, the default is LEFT.

Justification occurs if the initial value of a field is shorter than the length of the field as described in the panel body. Normally, right justification should be used only with output fields, since a right justified input field would be difficult to overtype.

For left or right, the justification applies only to how the field appears on the screen; leading blanks are automatically deleted when the field is processed. For asis, leading blanks are not deleted when the field is processed, nor when it is initialized. Trailing blanks are automatically deleted when a field is processed, regardless of its justification.

PAD(value)

Defines the pad character for initializing the field, and is valid only for input and output fields. The "value" may be:

NULLS
Any character, including blank (' ').

If this keyword parameter is omitted, the default is user-defined for input fields and blank for output fields.

If the field is initialized to blanks (or the corresponding dialog variable is null), the entire field will contain the pad character when the panel is first displayed. If the field is initialized with a value, the remaining field positions (if any) will contain the pad character.

Padding and justification work together in the following manner. On initialization, the field is justified (unless asis was specified) and then padded. For left justified and asis fields, the padding will extend to the right. For right justified fields, the padding will extend to the left.

The pad characters are automatically deleted when the field is processed.

Passing Attributes from Dialog Variables

In the above discussion of attribute keywords, the "value" is always shown as a literal. The value may also be expressed as a dialog variable name, preceded by an ampersand (&). Example:

```
INTENS(&A)
```

Variable substitution is done after processing of the initialization section.

The current value of the dialog variable must be valid for the particular keyword. In the above example, the value of dialog variable A must be HIGH, LOW, or NON.

Exception: TYPE(TEXT) must be coded explicitly. That is,

```
TYPE(&A)
```

is invalid if the current value of dialog variable A is TEXT.

Default Attribute Characters

The following default attribute characters are provided:

```
%  TYPE(TEXT) INTENS(HIGH)
+  TYPE(TEXT) INTENS(LOW)
_  TYPE(INPUT) INTENS(HIGH) CAPS(ON) JUST(LEFT)
```

If additional kinds of fields are required, an attribute section must be used to define additional attribute characters (or to change the attributes for any of the default characters).

In addition, the three default characters may be changed by means of a keyword on either the)ATTR or)BODY header statement. Format:

```
DEFAULT(abc)
```

where "a", "b", and "c" are the three characters to take the place of "%", "+", and "_" respectively.

Note: The value inside the parentheses must consist of exactly three characters, not enclosed in apostrophes and not separated by commas or blanks.

Typically, this keyword would be used on the)ATTR header statement if the three default characters are to be changed, and additional attribute characters are also to be defined. The keyword would be used on the)BODY header statement (and the entire attribute section would be omitted) if the only change is to redefine the default characters. Example:

```
)ATTR  DEFAULT($¢_)
   ¬  TYPE(INPUT) INTENS(NON)
   #  TYPE(OUTPUT) INTENS(LOW) JUST(RIGHT) PAD(0)
)BODY
```

In this example, the default characters for text fields are changed to "$" for high intensity, and "¢" for low intensity. The default character for high intensity input fields is specified as "_" (unchanged from the SPF-supplied default). Two additional attribute characters are then defined: "¬" for non display input fields, and "#" for low intensity output fields. The output fields are to be right justified and padded with zeros, presumably for displaying numeric data with leading zeros.

PROCESSING CONSIDERATIONS

When the DISPLAY service is invoked from a dialog function, any or all of the following parameters may be specified: panel name, message id, cursor field. The following processing occurs:

1. If a panel name has been specified, and a message id has not been specified, the panel is displayed without a message.

2. If both a panel name and a message id have been specified, the panel is displayed with an initial message (typically, a prompt or confirmation message).

3. If a message id has been specified, but a panel name has not been specified, the previously displayed panel is redisplayed with the message (typically, an error message).

4. If neither a panel name nor a message id has been specified, the previously displayed panel is redisplayed.

In the first two cases, processing of the panel definition proceeds normally, through the)INIT section, prior to display of the panel. If .MSG or .CURSOR is set in the)INIT section, that setting will override an initial message or cursor position passed via the calling sequence parameters.

In cases three and four, processing of the)INIT section will be bypassed, and there will be no automatic initialization of variables in the panel body (nor in the attribute section). As a result, all variables in the panel body will appear as last displayed, and input fields will contain whatever the user last entered. If an initial message or cursor position is passed via the calling sequence parameters, that setting will be used since the)INIT section is bypassed.

After the panel has been displayed, the user may enter information and press the ENTER key. All input fields are automatically stored into dialog variables of the same name, and the)PROC section of the panel definition is then processed. If any condition occurs that causes a message to be displayed (verification failure, MSG=value condition in a TRANS, or explicit setting of .MSG), processing continues to the)END statement. The panel is then redisplayed with the first (or only) message that was encountered.

When the user again presses ENTER, all input fields are stored and the)PROC section is again processed. This sequence continues until the entire)PROC section has been processed without any message conditions encountered. The panel display service then returns to the dialog function that invoked it with a return code of 0.

Whenever a panel is displayed or redisplayed, the user may press the End PF key. When End is pressed, all input fields are stored and the)PROC section is processed but no message is displayed (even if a MSG condition is encountered). The panel display service then returns to the dialog function with a return code of 8.

SYNTAX RULES AND RESTRICTIONS

General

- All statements, variable names, and keywords must be coded in uppercase. Values that are interpreted by the DISPLAY service, such as INTENS(LOW), must also be in uppercase. Values assigned to dialog variables and text in the panel body need not be in uppercase.

- All header statements,)ATTR,)BODY, etc., must be coded exactly as shown starting in column 1. Statements following the header need not begin in column 1.
- If a section is omitted, the corresponding header statement should also be omitted. The)BODY header may be omitted if the entire attribute section is omitted and there is no need to override the default attribute bytes via a keyword on the)BODY statement.
- An)END statement is required as the last line of each panel definition.

Blanks and Comments

- Blank lines may occur anywhere within the attribute, initialization, and processing sections.
- In the attribute section, the attribute character and all keywords that follow must be separated by one or more blanks. At least one keyword must follow the attribute character on the same line. Keywords may be continued on succeeding lines.
- In the initialization and processing sections, multiple statements may occur on the same line, separated by one or more blanks. Statements may not be split between lines, except that listed items within parentheses may be continued on succeeding lines (see below).
- One or more blanks may optionally occur on either side of an equal sign (=) or a not-equal operator (¬=). Embedded blanks may not occur in the not-equal operator ("¬ =" is invalid).
- One or more blanks may optionally occur on either side of parentheses (except that a blank may not follow the right parenthesis that begins a header statement). The following are all equivalent:

  ```
  INTENS(LOW)
  INTENS (LOW)
  INTENS ( LOW )
  ```

  Note: One or more blanks must follow the closing parenthesis to separate it from the next statement or keyword.
- Comments may be coded in the attribute, initialization, and processing sections. Comments must be enclosed with the PL/I comment delimiters, /* and */. The comment must be the last item on the line (i.e., additional keywords or statements may not follow the comment on the same line). A comment may not be continued on the next line. For multi-line comments, the comment delimiters must be used on each line.

Listed Items

- Listed items within parentheses may be separated by commas and/or one or more blanks. This includes paired values within a TRANS. The following, for example, are all equivalent:

  ```
  TRANS (&XYZ 1,A 2,B 3,C MSG=xxxx)
  TRANS (&XYZ 1 A 2 B 3 C MSG=xxxx)
  TRANS (&XYZ, 1 , A , 2 , B , 3 , C , MSG=xxxx)
  ```
- Null items within a list are treated the same as blank items. The following, for example, are equivalent:

  ```
  TRANS (&XXX N,' ',  Y,YES,  *,' ')
  TRANS (&XXX N,,     Y,YES,  *,)
  ```

- Listed items within parentheses may be continued on one or more lines. Example:

```
TRANS (&CASE 1,'THIS IS THE VALUE FOR CASE 1'
             2,'THIS IS THE VALUE FOR CASE 2')
```

  Literal values within a list may be split between lines by coding a plus sign (+) as the last character on each line that is to be continued. Example:

```
TRANS (&CASE 1,'THIS IS THE VALUE   +
      FOR CASE 1'  2,'THIS IS THE   +
      VALUE FOR CASE 2')
```

Variables within Text Fields and Literals

- In the panel body, a variable may appear within a text field. In the initialization and processing sections, a variable may appear within a literal value. In both cases, the variable name (and the preceding ampersand) are replaced with the value of the corresponding dialog variable. For example, if variable V has the value ABC then:

```
'F &V G'  yields  'F ABC G'
'F,&V,G'  yields  'F,ABC,G'
```

  Note that any non-alphameric character may terminate the variable name, such as a comma in the second example above.

- A period (.) at the end of a variable name has a special meaning. It causes concatenation with the character string following the variable. Example:

```
'&V.DEF'  yields  'ABCDEF'
```

- A single ampersand followed by a blank is interpreted as a literal ampersand character (not the beginning of a substitutable variable). An ampersand followed by a non-blank is interpreted as the beginning of a substitutable variable.

- A double ampersand may be used to produce a character string starting with (or containing) an ampersand. The double character rule also applies to apostrophes within literal values (if the literal is enclosed within delimiting apostrophes), and to a period if it immediately follows a variable name. That is:

```
&&  yields  &
''  yields  '  within delimiting apostrophes
..  yields  .  immediately following a variable name.
```

- When variable substitution occurs within a text field in the panel body, left or right shifting extends to the end of the field (defined by the occurrence of the next attribute byte). For left shifting, the rightmost character in the field is replicated (shifted in), provided it is a special (non-alphameric) character. Example:

```
%DATA SET NAME: &DSNAME ---------------------%
```

  Assuming that the value of variable DSNAME is greated than seven characters, the dashes will be "pushed" to the right, up to the next start of field (the next "%" in this example). If the value of DSNAME is less than seven characters, additional dashes will be "pulled" in from the right.

PANEL DEFINITIONS - FORMATTING GUIDELINES

In any panel definition, the first three lines include system-defined areas for the display of messages, and may include a primary input field and a scroll field.

Specific requirements are as follows:

- Short message area. The use of short messages is optional (see section entitled "Message Definitions"). If they are used, they are always displayed at the right-hand end of the first line of the panel body. They are first truncated to 24 characters, and then right justified.

  Short messages temporarily overlay whatever information is currently displayed in the right-hand end of the first line, and are automatically removed from display on the next interaction. (The original information is redisplayed when the message is removed.)

- Long message area. Long messages are always displayed in the third line of the panel body. As with short messages, they temporarily overlay whatever information is currently displayed on that line, and are removed from display on the next interraction.

- Primary input field. The primary input field is defined as the first input field in the panel body. If the user equates a PF key to an application-defined command and then presses that key, it will appear to the dialog function as if the user had typed the command in the primary input field and then pressed the ENTER key.

  Use of application-defined commands is optional. The first input field on a panel has special significance only when an application-defined command is equated to a PF key.

- Scroll field. When scrollable data is displayed (browse, edit, and table display), the scroll amount field must be the second input field in the panel definition, and must be exactly 4 characters in length.

Following are suggestions for formatting the first three lines of a panel body:

line 1	Title	Short Message
line 2	Primary Input or Prompt	Scroll
line 3	Long Message	

Line 1 should contain a title indicating the function being performed or, where appropriate, should display information critical to that function. The right-hand 24 characters of line 1 should not contain critical information if short messages are to be used.

If short messages are used, they should provide a brief indicatation of:

- Successful completion of a processing function, or
- Error conditions (accompanied by audible alarm).

Short messages should either be used consistently throughout the application, or not at all.

For table display, the short message area always contains an indication of current row/column positions, except when overlaid by a function-requested message. The row/column indication is automatically generated by the TBDISPL service, and replaces whatever was in the panel definition in that area.

For panels that allow application-defined commands, the primary input field should be on line 2. This same area should be used for the option entry field on selection menus. For panels that do not allow application-defined commands, line 2 should contain a prompt or other information that is significant to the user.

For table display panels, the scroll field should be at the right-hand end of line 2, following the primary input area. A scroll field is not meaningful for other types of panels, and should be omitted.

Line 3 should generally be left blank, so that long messages will not overlay any significant information. An exception to this rule might be made in the case of table display panels, to allow as much scrollable data as possible to fit on the screen.

Following are additional suggestions for designing panels with good human factors:

- Avoid overly cluttered panels. Split up "busy" panels into two or more simple panels that have less information and are easier to read.

- Do not use the last available line in a panel body. For example, if the dialog may be used on 24 line terminals, limit the body to 23 lines or less. The reason for this is that in split screen mode the maximum length of a logical screen is one less than the length of the physical screen.

- Place important input fields near the top of the panel and less important fields (especially optional input fields) further down, for two reasons: It is easier to move the cursor down than up, and in split screen mode the bottom of the panel may not be visible unless the user repositions the split line.

- Where practical, align fields vertically on a panel, especially input fields. Group related input fields under a common heading. Minimize use of multiple input fields on the same line, so that the NEW LINE key may be used to skip from one input field to the next.

- Use visual signals to indicate particular types of fields, such as arrows to indicate input fields, and colons to indicate variable information that is protected. Examples:

  ```
  SELECT OPTION ===>

  EMPLOYEE SERIAL:  123456
  ```

  In any case be consistent. Arrows, colons, and other visual signals are very confusing if used inconsistently.

- Use highlighting sparingly. Too many intensified fields result in visual confusion. Again, be consistent. Highlight the same type of information on all panels.

PANEL DEFINITIONS - SPECIAL REQUIREMENTS

This section describes special requirements for defining selection menus, tutorial pages, and table display panels.

SELECTION MENUS

A selection menu is a special type of panel that is processed by the SELECT service. A selection menu must have an input field named OPT. It must also have a processing section in which variable OPT is truncated at the first period and then translated to a character string. The results must be stored in a variable named SEL (see below).

A selection menu may have additional input fields, besides OPT, to set up dialog variables needed by the particular application. Any variables other than OPT and SEL that are set from a selection menu are automatically stored in the shared variable pool.

Variables from the shared pool (including system variables) may also be displayed on a selection menu to provide information to the end user.

The general format of the processing section of a selection menu is as follows:

```
)PROC
  &SEL = TRANS( TRUNC(&OPT,'.')
                value, 'string'
                value, 'string'
                  .
                  .
                  .
                value, 'string'
                  ' ', ' '
                   *, '?'       )
```

Each "value" is one of the options that may be entered on the menu. Each "string" contains selection keywords indicating the action to occur. The selection keywords that may be specified are:

```
 /  PANEL(name)    [NEWAPPL/NEWPOOL]                                 \
 |  CMD(command)   [NEWAPPL/NEWPOOL]                                 |
 |  PGM(program-name)  [PARM(parameters)]  [NEWAPPL/NEWPOOL]         |
 \  EXIT                                                             /
```

The selection keywords have the same meaning as for the SELECT service. The PANEL keyword, for example, is used to specify the name of a lower level selection menu to be displayed. The CMD or PGM keyword is used to invoke a dialog function coded in a command language or programming lanugage, respectively. Note that the OPT keyword (which is valid for the SELECT service) is not valid on a selection menu.

The EXIT keyword, if used, applies only to a primary option menu. It may be used to terminate SPF using defaults for list/log file processing.

Except for EXIT, each string of keywords must be enclosed in apostrophes, since it contains parentheses (and sometimes blanks).

If no option is entered (OPT variable is blank), a blank should be returned as the translated string. This will cause the SELECT service to redisplay the menu.

If an invalid option is entered (indicated by an asterisk, meaning none of the above), a question mark (?) must be returned as the translated string. This will cause the SELECT service to redisplay the menu with an "invalid option" message.

The reason for the truncation of the OPT variable prior to translation is to allow the end user to bypass intermediate menus. For example, "3.1" means primary option 3, suboption 1. Only the next lower menu is specified via the PANEL keyword. When the SELECT service discovers that variable OPT (which was automatically stored, untranslated, as the user entered it) contains a period, it will cause the next lower level menu to be selected with an initial option of everything following the first period. As long as the initial option is non-blank, the lower level menu will be processed in the normal fashion but not displayed to the end user.

Primary Option Menus

An example of a primary option menu is shown in Figure 18. This is the primary option menu for the SPF program development facility. The required input field, named OPT, appears in the second line of the panel body. It is followed by a description of the various options avaliable to the user.

This menu also has four variables within text fields at the upper right hand part of the screen. These reference system variables (from the shared variable pool) to display user id, time, terminal type, and number of PF keys.

The initialization section sets the control variable .HELP to the name of a tutorial page, to be displayed if the user presses the Help PF key from this menu. It also initializes two system variables that specify the tutorial table of contents and first index page. See discussion under "Help/Tutorial Pages."

The processing section specifies the action to be taken for each option entered by the user. If option 0 is selected, panel ISPOPT (a lower level selection menu) will be displayed. If option 1 is selected, program ISPBRO will be invoked. And so on.

Other applications may wish to include "SPF parms" and tutorial (and possibly other options from the SPF program development facility) on a primary option menu. They need not be invoked with the same selection codes, but the keywords that are returned in variable SEL should be the same as for the SPF primary option menu.

Note that for the tutorial, program ISPTUTOR is invoked and passed a parameter (T), which ISPTUTOR interprets as the name of the first panel to be displayed. Panel T is the first panel in the tutorial for the SPF program development facility. Other applications should pass the name of the first tutorial page for that application.

```
%----------------------- SPF-MVS PRIMARY OPTION MENU ------------------------
%SELECT OPTION ===>_OPT     %
%                                                              +USERID   - &ZUSER
%   0 +SPF PARMS  - SPECIFY TERMINAL AND SPF PARAMETERS        +TIME     - &ZTIME
%   1 +BROWSE     - DISPLAY SOURCE DATA OR OUTPUT LISTINGS     +TERMINAL - &ZTERM
%   2 +EDIT       - CREATE OR CHANGE SOURCE DATA               +PF KEYS  - &ZKEYS
%   3 +UTILITIES  - PERFORM SPF UTILITY FUNCTIONS
%   4 +FOREGROUND - COMPILE, ASSEMBLE, LINK EDIT, OR DEBUG
%   5 +BACKGROUND - COMPILE, ASSEMBLE, OR LINK EDIT
%   6 +COMMAND    - ENTER TSO COMMAND OR CLIST
%   7 +SUPPORT    - TEST DIALOG OR CONVERT MENU/MESSAGE FORMATS
%   T +TUTORIAL   - DISPLAY INFORMATION ABOUT SPF
%   X +EXIT       - TERMINATE SPF USING LIST/LOG DEFAULTS
%
+PRESS%END KEY+TO TERMINATE SPF
%
)INIT
  .HELP = TTUTOR
  &ZHTOP = TTUTOR       /* TUTORIAL TABLE OF CONTENTS */
  &ZHINDEX = TINDEX     /* TUTORIAL INDEX - 1ST PAGE  */
)PROC
  &SEL = TRANS( TRUNC (&OPT,'.')
                0,'PANEL(ISPOPT)'
                1,'PGM(ISPBRO)'
                2,'PGM(ISPEDIT)'
                3,'PANEL(ISPUTIL)'
                4,'PANEL(ISPFORA)'
                5,'PANEL(ISPJOB)'
                6,'PGM(ISPTSO)'
                7,'PANEL(ISPQTAC) NEWPOOL'
                T,'PGM(ISPTUTOR) PARM(T)'
              ' ',' '
                X,'EXIT'
                *,'?' )
)END
```

Figure 18. SPF Primary Option Menu

Master Application Menu

A master application menu, named ISP@MSTR, is distributed with SPF as part of the panel library. This menu may be used, if desired, to allow selection of the various applications available at an installation.

If used, the master menu should be the first menu displayed when the user logs on. It may be displayed automatically by including the following command in the user's TSO LOGON procedure or CMS PROFILE EXEC:

```
ISPF PANEL(ISP@MSTR)
```

The master menu, as distributed, is shown in Figure 19.

The distributed version of the master menu has only three options. Option "1" causes the primary option menu for the SPF program development facility to be displayed. Options "P" and "X" provide access to the SPF parms and exit functions directly from the master menu.

To add a new application to the master menu, a line should be added to the panel body, indicating the selection code and the nature of the application. A corresponding addition must then be made the the)PROC section, to specify the selection keywords for the option.

```
%------------------------  MASTER APPLICATION MENU  ---------------------------
%SELECT APPLICATION ===>_OPT      +
%                                                              +USERID   - &ZUSER
%                                                              +TIME     - &ZTIME
%   1 +SPF        - SPF PROGRAM DEVELOPMENT FACILITY           +TERMINAL - &ZTERM
%                                                              +PF KEYS  - &ZKEYS
%
%
%
%
%
%
%
%
%
%
%
%   P +PARMS      - SPECIFY TERMINAL PARAMETERS AND LIST/LOG DEFAULTS
%   X +EXIT       - TERMINATE USING LIST/LOG DEFAULTS
%
+PRESS%END KEY+TO TERMINATE
%
)INIT
)PROC
  &SEL = TRANS( TRUNC (&OPT,'.')
                1,'PANEL(ISP@PRIM) NEWAPPL'
            /*                          */
            /* ADD OTHER APPLICATIONS HERE */
            /*                          */
                P,'PANEL(ISPOPT)'
                X,'EXIT'
              ' ',' '
                *,'?' )
)END
```

Figure 19. Master Application Menu

Lower Level Selection Menus

Lower level selection menus follow the same rules as for a master or primary option menu. The SPF primary option menu is itself a lower level menu when invoked from the master menu.

Another example of a lower level menu is shown in Figure 20. This is the MVS version of panel ISPUTIL, which is displayed if option 3 is selected from the SPF primary option menu. For option 1, it specifies that program ISPUDA is to receive control, and that ISPUDA is to be passed a parameter (UDA1) which ISPUDA interprets as the name of a panel to be displayed.

An exit option is not included on this menu, since it is never displayed as a primary option menu.

Note: In this menu, variable OPT need not have been truncated prior to translation, since there are no lower level selection menus that can be displayed from this menu.

```
%---------------------- UTILITY SELECTION MENU -----------------------------
%SELECT OPTION ===>_OPT     +
%
%   1 +LIBRARY   - LIBRARY UTILITY:
+                              PRINT INDEX LISTING OR ENTIRE DATASET
+                              PRINT, RENAME, DELETE, OR BROWSE MEMBERS
+                              COMPRESS DATASET
%   2 +DATASET   - DATASET UTILITY:
+                              DISPLAY DATASET INFORMATION
+                              ALLOCATE, RENAME, OR DELETE ENTIRE DATASET
+                              CATALOG OR UNCATALOG DATASET
%   3 +MOVE/COPY - MOVE OR COPY MEMBERS OR DATASETS
%   4 +CATALOG   - CATALOG MANAGEMENT:
+                              DISPLAY OR PRINT CATALOG ENTRIES
+                              INITIALIZE OR DELETE USER CATALOG ALIAS
%   5 +RESET     - RESET STATISTICS FOR MEMBERS OF SPF LIBRARY
%   6 +HARDCOPY  - INITIATE HARDCOPY OUTPUT
%   7 +VTOC      - DISPLAY OR PRINT VTOC ENTRIES FOR A DASD VOLUME
%   8 +OUTLIST   - DISPLAY, DELETE, OR PRINT HELD JOB OUTPUT
%   9 +SCRIPT/VS - FORMAT, DISPLAY, AND OPTIONALLY PRINT SCRIPT TEXT
)INIT
  .HELP = TU
)PROC
  &SEL = TRANS( TRUNC (&OPT,'.')
               1,'PGM(ISPUDA) PARM(UDA1)'
               2,'PGM(ISPUDA) PARM(UDA2)'
               3,'PGM(ISPUMC)'
               4,'PGM(ISPUCA)'
               5,'PGM(ISPURS)'
               6,'PGM(ISPUHC)'
               7,'PGM(ISPUVT)'
               8,'PGM(ISPUOL) PARM(UOL01)'
               9,'PGM(ISPUSC) PARM(SCRPTA)'
             ' ',' '
               *,'?'  )
)END
```

Figure 20. Lower Level Selection Menu

A help or tutorial page is a special type of panel that is processed by the SPF tutorial program. (The tutorial program invokes the panel display service to display the panel.) The tutorial program may be invoked either from a selection menu, or via the Help PF key.

Tutorial panels are arranged in a hierarchy. When the tutorial is entered from a selection menu, the first panel to be displayed is normally the top of the hierarchy. The name of the first panel is passed as a parameter to the ISPTUTOR program (see discussion of primary option menus).

When the tutorial is entered via the Help PF key, the first panel to be displayed is some appropriate panel within the hierarchy, depending upon what the user was doing when help was requested. In this case, the name of the panel is specified by the .HELP control variable in a panel or message definition.

When viewing the tutorial, the user may select topics by entering a selection code, or simply press the ENTER key to view the next topic. On any panel, the user may also enter the following commands:

```
BACK  or B - to back up to the previously viewed panel
SKIP  or S - to skip to the next topic
UP    or U - to display a higher level list of topics
TOP   or T - to display the table of contents
INDEX or I - to display the tutorial index.
```

The name of the top panel must be specified by dialog variable ZHTOP, and the name of the first index panel must be specified by ZHINDEX. It is recommended that these two dialog variables be initialized at the beginning of the application to ensure that the end user can always display the tutorial top or index, regardless of how the tutorial was entered. One way to initialize these variables is to set them from the primary option menu. For an example, see Figure 18.

Each tutorial panel must have a "next selection" input field named OPT. It should also have a processing section in which the following variables are set:

SEL	Specifies the name of the next panel to be displayed based on the topic selected by the user (by translating OPT to a panel name). The panel name may be preceded by an asterisk (*) to indicate a topic that can be explicitly selected by the user, but which will be bypassed if the user presses the ENTER key to view the next topic. If this panel does not have any selectable topics, SEL should be omitted.
UP	Specifies the name of the parent panel, from which this panel was selected. Generally, UP may be omitted since the tutorial program remembers the sequence of selections that lead to the display of this panel. UP is used only if this panel is the first to be displayed (via the Help key) or is selected from the tutorial index, and the user then enters the UP command.
CONT	Specifies the name of the next continuation panel. If there is no continuation panel, CONT should be omitted.

The entire processing section should be omitted if all of the variables SEL, UP, and CONT are omitted.

A panel cannot have both a continuation panel and selectable topics. However, the last panel in a sequence of continuation panels may have selectable topics.

Figure 21 shows a sample hierarchy of tutorial panels. Panels A and B each have three selectable topics. Panels C and D2 each have two selectable topics. The other panels have no selectable topics. Panel D1 has a continuation page (D2), and panel F1 has two continuation pages (F2 and F3).

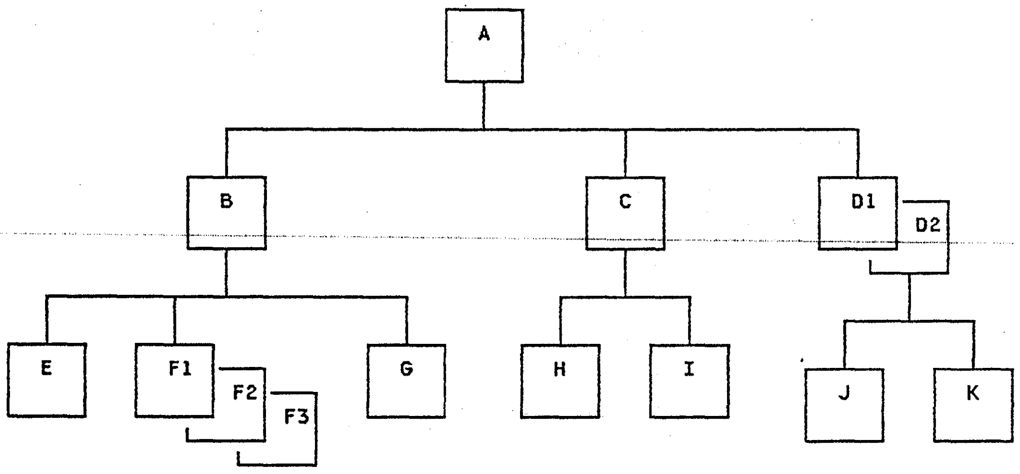


Figure 21. Sample Tutorial Hierarchy

Two sample tutorial panels are shown in Figure 22 and Figure 23. These are assumed to be panels B and F2 in the hierarchy.

Panel B has three selectable topics. In the processing section, OPT is translated to a panel name (E, F1, or G) corresponding to the selected option, and the result is stored in SEL. If none of the valid options is selected, a question mark (?) is returned as the translated string. This will cause the tutorial program to display an "invalid option" message.

Note that option 3 is translated to "*G". This indicates that panel G will be displayed if the user selects option 3, but will be bypassed if the user repeatedly presses the ENTER key to view each topic. (The order in which topics are presented when the ENTER key is pressed is the same as the order in which they appear in the TRANS function.)

In panel B, the name of the parent panel (A) is stored in variable UP.

Panel F2 has no selectable topics, but does have a continuation page. The name of the continuation panel (F3) is stored in variable CONT. The name of the parent panel (B) could have been stored in UP, but this was omitted assuming that F2 cannot be directly entered via the Help PF key or from the tutorial index.

```
%TUTORIAL ------------------- 3270 DISPLAY TERMINAL --------------------TUTORIAL
%NEXT SELECTION ===>_OPT     +

%                           -----------------------------------
                            |       GENERAL INFORMATION        |
                            |         3270 KEY USAGE           |
                            -----------------------------------
+
   THE IBM 3270 DISPLAY TERMINAL HAS SEVERAL KEYS WHICH WILL ASSIST YOU
   IN ENTERING INFORMATION.  THESE ARE HARDWARE DEFINED KEYS; THEY DO NOT
   CAUSE A PROGRAM INTERRUPTION.

   THE FOLLOWING TOPICS ARE PRESENTED IN SEQUENCE,
   OR MAY BE SELECTED BY NUMBER:

     %1+ INSERT AND DELETE KEYS
     %2+ ERASE EOF (TO END-OF-FIELD) KEY

   THE FOLLOWING TOPIC WILL BE PRESENTED ONLY IF
   EXPLICITLY SELECTED BY NUMBER:

     %3+ NEW LINE AND TAB KEYS

)PROC
  &SEL = TRANS(&OPT  1,E  2,F1  3,*G  *,'?')
  &UP  = A
)END
```

Figure 22. Sample Tutorial Panel (B)

```
%TUTORIAL -------------------- ERASE EOF KEY ------------------- TUTORIAL
%NEXT SELECTION ===>_OPT     +

+
   WHEN THE ERASE EOF (ERASE TO END OF FIELD) KEY IS USED, IT WILL APPEAR
   TO BLANK OUT THE FIELD.  ACUTALLY, NULL CHARACTERS ARE USED IN ERASING
   TO THE NEXT ATTRIBUTE BYTE, THUS MAKING IT EASY TO USE THE INSERT MODE
   (WHICH REQUIRES NULL CHARACTERS).

   IF THE ERASE EOF KEY IS PRESSED WHEN THE CURSOR IS NOT WITHIN A INPUT
   FIELD, THE KEYBOARD WILL LOCK UP.  PRESS THE RESET KEY TO UNLOCK THE
   KEYBOARD.

   YOU CAN TRY OUT THE ERASE EOF KEY BY ENTERING DATA ON LINE 2, THEN
   MOVING THE CURSOR BACK OVER PART OR ALL OF THE DATA AND PRESSING THE
   KEY.

                          (CONTINUED ON NEXT PAGE)

)PROC
  &CONT = F3
)END
```

Figure 23. Sample Tutorial Panel (F2)

TABLE DISPLAY PANELS

A table display panel is a special type of panel that is processed by the TBDISPL service. The panel definition contains non-scrollable text, including column headings, followed by a model line that defines the format for each line of the scrollable data. Attribute characters in the model line indicate whether each column is protected or unprotected (user-modifiable).

Typically, the left-most column in each line of scrollable data is defined as an unprotected selection field. A code entered in that field is interpreted by the dialog function to determine the particular processing for that row.

Specific requirements for each section of the panel definition are described in the following paragraphs.

- Attribute Section (Typically Required)

  Attribute characters may be defined for use in the panel body and the model line. For the model line, only the attributes TYPE, INTENS, and PAD are meaningful; all fields in the model line will assume CAPS(OFF) and JUST(LEFT).

  Typically, an attribute section is required since the model line usually contains output fields. There is no default attribute character for output fields.

- Body (Required)

  The panel body contains the non-scrollable text. It must also contain two, and only two, input fields:

  1. Command field - must be the first input field, and must be at least 8 characters long. The field may have any desired name. The user may enter a temporary scroll amount in this field and then press a scroll PF key (see description of scrolling in the SPF Program Reference Manual).

     This field may also be used for application-defined commands. The contents of the field are automatically stored into the corresponding dialog variable. Upon return from TBDISPL, the dialog function may interpret this field and take appropriate action.

  2. Scroll amount field - must be the second input field, and must be exactly 4 characters long. The field may have any desired name. Its initial value may be set in the)INIT section of the panel definition to any valid scroll amount.

  If additional input fields are specified in the panel body, they are ignored (may not be used to enter data).

- Model Section (required)

  The panel body must be followed by a)MODEL header statement, starting in column one. The)MODEL header is immediately followed by a single line, called the model line.

  The model line contains input and/or output fields, consisting of an attribute character immediately followed by the letter Z (the name of the null system variable). Only Z may be used as a variable name in the model line.

  The actual variable names that correspond to each input or output field in the model line are specified in the initialization section (see below).

  Text fields may also occur in the model line. A text attribute character may appear by itself to terminate the preceding input or output field.

Any characters that appear within a text field in the model line will be replicated in each line of the scrollable data. (This includes the letter Z; it is not treated as a variable name if it occurs in a text field.)

Variables within text fields (e.g., "+&XYZ") are not allowed in the model line; results are unpredictable.

- Initialization Section (Required)

  The initialization section must assign a name list (enclosed in parentheses) to the variable VARS, unless VARS is set by the dialog function before invoking TBDISPL. General format:

  ```
  )INIT
    &VARS = '(name1 name2 ... )'
  ```

  Each name in the list specifies the actual variable name represented by a null variable (Z) in the model line. The first name corresponds to the first Z, the second name to the second Z, etc. Names within the list must be separated by one or more blanks.

  Typically, the first name in the list will specify the dialog variable into which a selection code (entered by the user) will be stored, and all remaining names will correspond to columns in the table. However, this arrangement is not required; any name in the list may or may not correspond to a column in the table.

  The list should not include the names of extension variables that appear in some, but not all, rows of the table; results are unpredictable.

  The initialization section may also contain any statement that is valid in an initialization section of a panel definition, except that the only control variable that may be set is .HELP.

- Processing Section (Omit)

  The panel should not contain a processing section; the results are unpredictable.

When the panel is displayed, the model line is replicated to the end of the logical screen. Each input or output field that has a corresponding column in the table is initialized with data from succeeding rows from the table. The first row displayed is the row pointed by the CRP when TBDISPL was entered.

Input or output fields in the model line that do not correspond to columns in the table are initialized with the current contents of the corresponding dialog variables (in all rows). If these fields are to be blank, the corresponding variables must be set to blanks or null prior to each call to TBDISPL.

The user may scroll the data up and down, and may enter information in the command field and/or the input fields in a row (one row at a time). Processing of input is described in the TBDISPL service description.

Figure 24 shows a sample panel definition for table display. Assuming that the current contents of the table are as shown in Figure 25, the resulting display is shown in Figure 26.

In this example, the select field (left-most column) does not correspond to a column in the table; it is used to return a selection code, entered by the user, in a variable named SELECT. The other variables in the VARS name list correspond to variables in the table. The example also illustrates the initialization of the scroll amount field to PAGE, and the specification of a corresponding help panel.

```
)ATTR
  @ TYPE(OUTPUT) INTENS(LOW)
)BODY
%---------------------------- EMPLOYEE LIST ---------------------------------
%COMMAND INPUT ===>_OPT                                      %SCROLL ===>_AMT +

+SELECT      ------ EMPLOYEE NAME -------          -- PHONE ---         EMPLOYEE
+ CODE       LAST          FIRST        MI         AREA NUMBER           SERIAL
)MODEL
  _Z+       @Z            @Z           @Z         @Z   @Z                @Z
)INIT
  &VARS = '(SELECT LNAME FNAME I PHA PHNUM EMPSER)'
  &AMT  = PAGE
  .HELP = PERS123
)END
```

Figure 24. Table Display Panel Definition

EMPSER	LNAME	FNAME	I	PHA	PHNUM
598304	Roberston	Richard	P	301	840-1224
172397	Smith	Susan	A	301	547-8465
813058	Russell	Charles	L	202	338-9557
395733	Adams	John	Q	202	477-1776
502774	Caruso	Vincent	J	914	294-1168

Figure 25. Current Contents of Table

```
---------------------------- EMPLOYEE LIST --------- LINE 000001 COL 001 080
COMMAND INPUT ===> _                                        SCROLL ===> PAGE

SELECT      ------ EMPLOYEE NAME -------          --- PHONE ---         EMPLOYEE
 CODE       LAST          FIRST        MI         AREA  NUMBER           SERIAL
            Roberston     Richard      P          301  840-1224          598304
            Smith         Susan        A          301  547-8465          172397
            Russell       Charles      L          202  338-9557          813058
            Adams         John         Q          202  477-1776          395733
            Caruso        Vincent      J          914  294-1168          502774
******************************* END OF DATA **********************************
```

Figure 26. Table as Displayed

MESSAGE DEFINITIONS

SPF message definitions are stored in a message library and displayed by means of the DISPLAY or TBDISPL service, or written to the SPF log file via the LOG service. Messages are created or changed by editing directly into the message library. The messages are interpreted during SPF execution; no compile or preprocessing step is required.

Each message is referrenced by message id. A message id may be 4 to 8 characters long, as follows:

- Prefix: 1 to 5 alphabetic characters (A-Z, #, $, or @)
- Number: 3 numeric characters (0-9)
- Suffix (optional): 1 alphabetic character

Note: If the prefix is 5 characters long, the suffix must be omitted so that the total length will not exceed 8 characters.

Several messages may be contained within each member of the message library. The member name is determined by truncating the message id after the second digit of the number. Examples:

Message id	Member name
G015	G01
ISPE241	ISPE24
XYZ123A	XYZ12
ABCDE965	ABCDE96

All messages which have ids beginning with the characters "G01", for example, must be in member G01. Within the member, the messages must appear in collating sequence by message id.

Each message consists of two lines, as follows:

```
msgid ['short message'] [.HELP = panel/*] [.ALARM = YES/NO]

'long message'
```

Specification of a short message is optional. If a short message is specified, it will be displayed first. Short messages are automatically right-justified and displayed at the right hand end of the first line on the screen. If the user presses the Help PF key, the long message will then be displayed on the third line of the screen. If the user presses the Help PF key again, tutorial mode will be entered.

If a short message is not specified, the long message will be displayed first, on the third line of the screen. If the user then presses the Help PF key, tutorial mode will be entered.

If tutoral mode is entered by the user, the panel name specified by .HELP will be the first tutorial page displayed. If .HELP=* is specified, the first tutorial page will be whatever was specified in the panel definition (i.e., the panel on which this message is being displayed). The default is "*" if .HELP is not specified.

If .ALARM=YES is specified, the audible alarm will be sounded whenever the message is displayed. If .ALARM=NO is specified, the alarm will not be sounded. The default is NO if .ALARM is not specified.

When messages are written to the SPF log file, both the short message (if any) and the long message are written in the same output line. The short message comes first, followed by the long message.

Substitutable parameters, consisting of a dialog variable name preceded by an ampersand (&), may appear anywhere within the short and long message text. Example:

```
    'VOLUME &VOL NOT MOUNTED'
```

Substitutable parameters may also be used to specify the value of .HELP or .ALARM, as follows:

```
    'VOLUME &VOL NOT MOUNTED'  .HELP = &H  .ALARM = &A
```

where variable H must contain a panel name or single asterisk, and variable A must contain YES or NO.

After substitution of the variables, the short message is truncated to 24 characters and the long message is truncated to 79 characters.

Syntax rules:

1. The message id must begin in column 1 of the first line, and the long message must begin in column 1 of the second line. For readability, one or more blank lines may separate the two-line message specifications within the member.

2. In the first line, the message id, short message, .HELP, and .ALARM fields must be separated by at least one blank. One or more blanks may optionally occur on either side of an equal sign (=).

3. The short message (if specified) and the long message must each be enclosed in apostrophes (').

4. Within the short or long message text, any non-alphameric character may terminate a variable name. Example:

   ```
       'ENTER &X, &Y, OR &Z'
   ```

   where a comma terminates the variable names X and Y.

5. A period (.) at the end of a variable name has a special meaning. It causes concatenation with the character string following the variable. For example, if the value of variable V is ABC then:

   ```
       '&V.DEF'  yields  'ABCDEF'
   ```

6. A single ampersand followed by a blank is interpreted as a literal ampersand character (not the beginning of a substitutable variable). An ampersand followed by a non-blank is interpreted as the beginning of a substitutable variable.

7. A double ampersand may be used to produce a character string starting with an ampersand. The double character rule also applies to apostrophes (within the delimiting apostrophes required for the short and long message text), and to a period if it immediately follows a variable name. That is:

   ```
       &&  yields  &
       ''  yields  '  within delimiting apostrophes
       ..  yields  .  immediately following a variable name.
   ```

Figure 27 shows an example of a member in the message library. This member contains all message ids which begin with "EMPX21".

```
EMPX210  'INVALID TYPE OF CHANGE'    .HELP=PERS033   .ALARM=YES
'TYPE OF CHANGE MUST BE NEW, UPDATE, OR DELETE.'

EMPX213  'ENTER FIRST NAME'          .HELP=PERS034   .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE = NEW OR UPDATE.'

EMPX214  'ENTER LAST NAME'           .HELP=PERS034   .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE = NEW OR UPDATE.'

EMPX215  'ENTER HOME ADDRESS'        .HELP=PERS035   .ALARM=YES
'HOME ADDRESS MUST BE ENTERED FOR TYPE OF CHANGE = NEW OR UPDATE.'

EMPX216  'AREA CODE INVALID'         .ALARM=YES
'AREA CODE &PHA IS NOT DEFINED.  PLEASE CHECK THE PHONE BOOK.'

EMPX217  '&EMPSER ADDED'
'EMPLOYEE &LNAME, &FNAME &I ADDED TO FILE.'

EMPX218  '&EMPSER UPDATED'
'RECORDS FOR &LNAME, &FNAME &I UPDATED.'

EMPX219  '&EMPSER DELETED'
'RECORDS FOR &LNAME, &FNAME &I DELETED.'
```

Figure 27. Sample Member in Message Library

SKELETON DEFINITIONS

SPF skeleton definitions are stored in a skeleton library and accessed by means of the SPF file tailoring services. Skeletons are created or changed by editing directly into the skeleton library. The skeletons are interpreted during SPF execution; no compile or preprocessing step is required.

Note: The SPF-distributed skeleton library also contains old format SPF "proc" members. The description of skeleton formats which follows applies only to new format skeletons used with file tailoring services.

There are two types of records that may appear in the skeleton file:

1. Data Records

   These are a continuous stream of intermixed text, variables, and control characters that are processed to create an output record.

2. Control Statements

   These control the file tailoring process. Control statements start with a right parenthesis ")" in column 1. Records containing a ")" in column 1, and a blank in column 2, are interpreted as data records. Records containing a ")" in column 1 and a non-blank character in column 2, are interpreted as control statements.

   Note: A)DEFAULT control statement can be used for assigning different special characters for syntactical purposes.

DATA RECORDS

Columns 1-71 of each data record are scanned and processed as described below. After variable substitution, the results are truncated (if required) to a length of 80 and copied to columns 1-80 of the output record.

If more than one input record maps to a single output record, continuation is specified by a question mark (?) in column 72 of each input record that is to be continued. If any character other than a question mark appears in column 72 of an input record, it is copied to column 72 of the output record. In this case, column 72 of the output record must not contain generated data (i.e., it must be blank) for the continuation character to be copied. Otherwise, a severe error results.

The following control characters have special meanings:

- An ampersand (&) indicates the start of a variable name. The value of the corresponding dialog variable is substituted in the output record. A value of all blanks is treated as null.

- The following characters implicitly delimit the end of a variable name:

  ```
  Blank ¢ < ( + | & ! * )
  ; ¬ - ? , % _ > : / ' = "
  ```

  This list includes the seven characters that may be overridden with the)DEFAULT control statement. If those characters are overridden, the specified characters are substituted in above list.

- A period (.) at the end of a variable name causes the value of the variable to be concatenated with the character string following the period. For example, if variable V has the value ABC then:

  ```
  "&V.DEF"  yields  "ABCDEF"
  ```

- An exclamation mark (!) is used as a tab character. It tabs the output record to the next tab stop and fills with blanks. The next character following exclamation mark in the input record is put at the tab stop location in the output record. Tab stops are specified via the)TB control statement.

- A less-than (<), vertical bar (|), and greater-than (>) symbol, respectively, specify the beginning, middle, and end of a conditional substitution string:

  ```
  <string1|string2>
  ```

  where "string1" must contain at least one variable name. "string2" can be null.

  If the first variable in "string1" is not null, "string1" is substituted in the output record. If the first variable in "string1" is null, "string2" is substituted in the output record.

Two consecutive control characters in the input record result in one control character being placed in the output record, i.e.

```
&&  yields  &
!!  yields  !
<<  yields  <
||  yields  |
>>  yields  >
..  yields  .  immediately following a variable name.
```

CONTROL STATEMENTS

The general format of a control statement, which must begin in column 1, is:

```
)Control-word   token1  ...  token31
```

where each token represents a name, value, operator, or keyword.

The tokens must be separated by one or more blanks, and may not contain embedded blanks. A token may be coded as:

- A character string,
- A dialog variable name, preceeded by an ampersand, or
- A concatenation of variable names and character strings.

The current value of each variable is substituted prior to evaluation of the control statement. The rules for delimiting a variable name and for the use of ampersands, periods, double ampersands, and double periods are the same as for data records. See description above.

Specific control statements are described below.

)DEFAULT abcdefg

The seven characters, represented by "abcdefg" override the use of the ")", "&", "?", "!", "<", "|", and ">" characters, respectively. Exactly seven characters must be specified, and they must be special (non-alphameric) characters.

The)DEFAULT statement takes affect immediately, when it is encountered. It retains affect until the end of FTINCL processing, or until another)DEFAULT statement is encountered.

)TB value1 ... value8

Up to 8 tab stops can be specified. A tab stop specifies a tab position in the output record, and must be in the range 1-80. The default is one tab stop at location 80.

)IM skel-name [NT] [OPT]

The specified skeleton is imbedded at the point where the)IM statement is encountered. Up to 3 levels of imbedding are permitted. The optional NT parameter indicates that no tailoring is to be performed on the imbedded skeleton.

The optional OPT parameter indicates that the skeleton may or may not be present. If the skeleton is not present, no error indication is given, and the record is ignored. If OPT is not coded, a severe error occurs if the skeleton is not present.

)SEL relational-expression

)ENDSEL

The relational expression is evaluated for a true or false condition. If the condition is true, the skeleton input records between the)SEL and the corresponding)ENDSEL are processed. If the condition is false, these records are skipped. Up to 8 levels of nesting are permitted.

The relational expression consists of a simple comparison of the form:

    value1 operator value2

or a combination of up to 8 simple comparisons joined by connectors. The system variable Z may be used to represent a null or blank value.

The allowable operators are:

EQ	or	=	LE	or	<=
NE	or	¬=	GE	or	>=
GT	or	>	NG	or	¬>
LT	or	<	NL	or	¬<

The allowable connectors are | (OR) and && (AND).

Examples:

```
)SEL  &COND = YES
)SEL  &TEST1 ¬= &Z  |  &ABC = 5
```

)DOT table-name

)ENDDOT

The skeleton input records between the)DOT and the corresponding)ENDDOT are iteratively processed, once for each row in the named table, beginning with the first row. At the start of each iteration, the contents of the current table row are retrieved (stored into the corresponding dialog variables). Those values can then be used as parameters in control statements or substituted into data records. Up to 4 levels of nesting are permitted. The same table cannot be processed recursively.

If the table was already open, it remains open after file tailoring with the CRP positioned at TOP. If it was not open, it is opened automatically and then closed upon completion of file tailoring.

)SET variable = expression

)SET allows a value to be assigned to a dialog variable. The variable name should <u>not</u> be preceded by an ampersand, unless the variable name is itself stored as a variable. The expression can be specified as either:

```
     value1
or:
     value1  operator  value2  operator  ...  value15
```

where "operator" can be a plus sign (+) or a minus sign (-).

)CM comment

The statement is treated as a comment. No tailoring is performed, and the record is not placed in the output file.

SAMPLE SKELETON FILE

A sample skeleton file is shown in Figure 28.

The sample skeleton references several dialog variables (ASMPARMS, ASMIN, MEMBER, etc.). It also illustrates use of select statements ")SEL" and ")ENDSEL" to conditionally include records. The first part of the example has nested selects to include concatenated macro libraries if the library names have been specified by the user (i.e., if variables ASMMAC1 and ASMMAC2 are not equal to the null variable Z).

In the second part of the example, select statements are used to conditionally execute a load-go step. An imbed statement, ")IM", is used to bring in a separate skeleton for the load-go step.

```
//ASM     EXEC  PGM=IFOX00,REGION=128K,
//              PARM=(&ASMPARMS)
//SYSIN     DD  DSN=&ASMIN(&MEMBER),DISP=SHR
//SYSLIB    DD  DSN=SYS1.MACLIB,DISP=SHR
)SEL  &ASMMAC1 ¬= &Z
//          DD  DSN=&ASMMAC1,DISP=SHR
)SEL  &ASMMAC2 ¬= &Z
//          DD  DSN=&ASMMAC2,DISP=SHR
)ENDSEL
)ENDSEL
//SYSUT1    DD  UNIT=SYSDA,SPACE=(CYL,(5,2))
//SYSUT2    DD  UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSUT3    DD  UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSPRINT DD   SYSOUT=(&ASMPRT)
)CM   IF USER SPECIFIED "GO", WRITE OUTPUT IN TEMP DATA SET
)CM   THEN IMBED "LINK AND GO" SKELETON
)SEL  &GOSTEP = YES
//SYSGO     DD  DSN=&&&&OBJSET,UNIT=SYSDA,SPACE=(CYL,(2,1)),
//              DISP=(MOD,PASS)
)IM   LINKGO
)ENDSEL
)CM   ELSE (NOGO), WRITE OUTPUT TO USER DATA SET
)SEL  &GOSTEP = NO
//SYSGO     DD  DSN=&ASMOUT(&MEMBER),DISP=OLD
)ENDSEL
//*
```

Figure 28. Sample Skeleton File

CHAPTER 6. DIALOG TESTING PROCEDURES

This chapter describes recommended set up procedures, test and trace modes, and use of the SUPPORT option for testing a dialog.

SET UP PROCEDURES

The following steps are recommended for development and testing of a dialog.

1. Before starting development, set up the panel, message, skeleton, table, and program libraries for the application. For ~~the MVS environment, this means allocating new partitioned~~ data sets. For the VM environment, this means selecting minidisks on which the libraries are to reside, and ensuring the dialog has access to the minidisks. See Chapter 3 for additional information on library setup procedures.

2. Create a command procedure (CLIST or EXEC2) that contains the necessary ALLOCATE or FILEDEF statements to allocate the libraries. The application libraries should be concatenated ahead of the libraries required by SPF, as described in Chapter 3. This command procedure should be executed prior to invoking the SPF program development facility, to ensure that the application libraries are accessible during testing. If desired, the command procedure may include an ISPF command to invoke the program development facility.

3. As a general rule, invoke the SPF program development facility in one of the test or trace modes prior to testing.

4. Create the panels, messages, and skeletons by editing directly into the application libraries. In the VM environment, these libraries can be updated only in test or trace mode. Use the SPF SUPPORT option (options 7.1 and 7.3) to display panels and messages as the end user will see them.

5. Create the dialog functions and assure that the text or load modules are in libraries (or on minidisks) accessible to SPF. See Chapter 3 for a discussion of program libraries. The functions may be tested by means of the SPF SUPPORT option (option 7.2).

   Note: Under MVS, functions coded as program modules must be link edited. Under VM, they may be link edited. In either environment, when a function is link edited the ISPLINK subroutine must be included (explicitly or via automatic call) in the load module. For MVS, ISPLINK is distributed in load module format and may be placed in a system library for automatic call during link edit. For VM, ISPLINK is distributed as a TEXT file.

6. Finally, invoke the application from the top (as the end user would do so) rather than testing pieces of it via the SUPPORT option. To do this, add an ISPF command to the command procedure created in step 2. In this case, the ISPF command should invoke the application, using the appropriate PANEL, CMD, or PGM parameter, rather than invoking the SPF program development facility. This command procedure may be made available to the end users as the means of invoking the application. Alternatives are to invoke the application from the master menu or other selection menu.

OPERATING IN TEST AND TRACE MODES

There are four mutually exclusive keyword parameters that may be specified on the ISPF command to control the operational mode:

- TEST - Test mode
- TESTX - Extended test mode
- TRACE - Trace mode
- TRACEX - Extended trace mode

In TEST mode, SPF operates differently from normal mode in the following ways:

1. Panel and message definitions are refetched from the panel and message libraries whenever a panel name or message id is specified in an SPF service. (In normal mode, the most recently accessed panel definitions are retained in virtual storage to reduce I/O operations and, under MVS, BLDL macros for frequently used panels and messages are issued during SPF initialization to reduce search time.) If you have modified the panel or message library, use of TEST mode will ensure that the latest version of each panel or message is accessed during a test run.

2. Tutorial panels are displayed with current panel name, previous panel name, and previous message id on the bottom line of the display screen. This will assist you in identifying the position of the panel in the tutorial hierarchy.

3. Screen printouts (obtained via the PRINT or PRINT-HI PF keys) include line numbers, current panel name, and message id.

4. If a dialog function is operating in the CANCEL error mode (which is the default), the panel that is displayed on an error allows you to force the dialog to continue, in spite of the error. Results from that point on may be unpredictable.

5. Other than the case discussed in item 4 above, any SPF-detected error, ABEND, or program interrupt forces an ABEND of all of SPF. The user may also force an ABEND by entering ABEND or CRASH in the command line of any panel.

6. MVS/TSO only:

   - The PA1 key causes an immediate exit out of SPF.

   - If an SPF subtask ABENDs, a dump may be taken by pressing ENTER after the ABEND message appears, provided that a SYSUDUMP, SYSMDUMP, or SYSABEND DD has been allocated.

7. VM/CMS only:

   - An ADSTOP set within SPF code will not be lost, even if SPF invokes a CMS command that executes in the user area. If SPF is operating in DCSS, the page containing the ADSTOP will be marked non-sharable, and will be copied automatically to the user area.

In TESTX (extended test) mode, SPF operates the same as in TEST mode except that all messages written to the SPF log file are also displayed at the terminal.

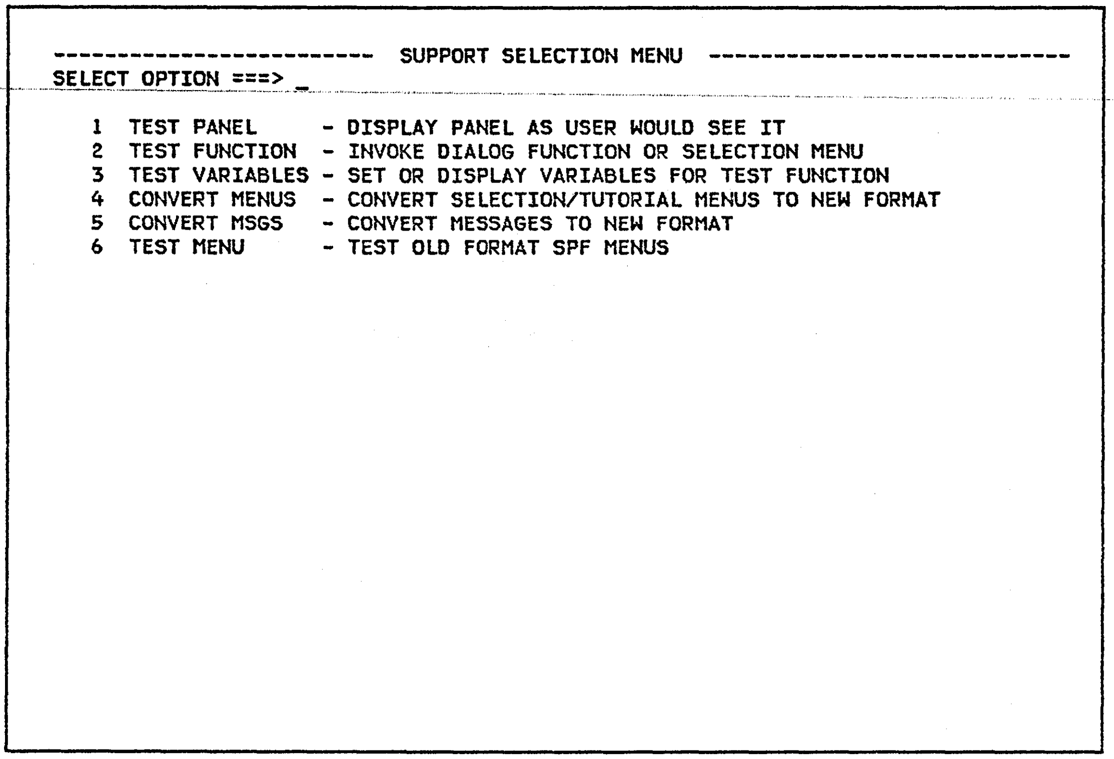
In TRACE mode, SPF operates the same as in TEST mode except that a message is written to the SPF log file whenever any SPF service is invoked from a CLIST or EXEC2, and whenever any error is detected by an SPF service (even if CONTROL ERRORS RETURN has been issued).

In TRACEX (extended trace) mode, SPF operates the same as in TRACE mode except that all messages written to the SPF log file (including the trace messages) are also displayed at the terminal.

SUPPORT (OPTION 7)

The support option is part of the SPF program development facility (primary option 7). It provides dialog test aids and conversion utilities. The support option is described here as well as in the SPF Program Reference manual.

The support selection menu is shown in Figure 29. This menu is displayed after option 7 is selected from the SPF primary option menu.



```
------------------------  SUPPORT SELECTION MENU  ---------------------------
SELECT OPTION ===> _

   1  TEST PANEL     - DISPLAY PANEL AS USER WOULD SEE IT
   2  TEST FUNCTION  - INVOKE DIALOG FUNCTION OR SELECTION MENU
   3  TEST VARIABLES - SET OR DISPLAY VARIABLES FOR TEST FUNCTION
   4  CONVERT MENUS  - CONVERT SELECTION/TUTORIAL MENUS TO NEW FORMAT
   5  CONVERT MSGS   - CONVERT MESSAGES TO NEW FORMAT
   6  TEST MENU      - TEST OLD FORMAT SPF MENUS
```

Figure 29. Support Selection Menu

A new shared variable pool, referred to as the "test pool," is established when option 7 is invoked. Any variables that are set or displayed by options 7.1 or 7.3 are from this pool. Functions that are invoked under option 7.2 may copy variables from and to the test pool by means of VGET and VPUT services.

The test pool simply takes the place of the normal shared variable pool to isolate variables in the option 7 environment from variables being used in the normal environment.

Installations that have previously extended or custom tailored SPF may need to convert old format selection menus to the new panel formats. A conversion utility to assist in this process is provided by options 7.4.

The following sections describe each of the support functions, corresponding to the six options on the support selection menu.

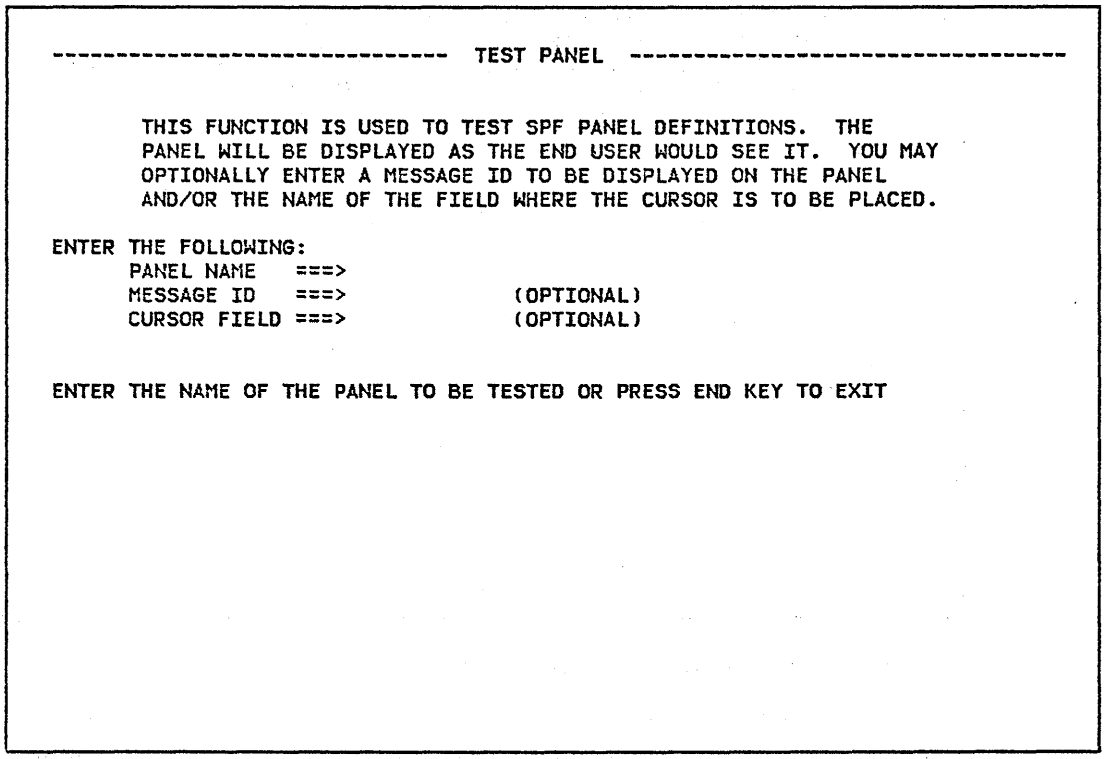
TEST PANEL (OPTION 7.1)

When this option is selected, a panel is displayed that allows entry of the name of a panel to be tested. A message id and initial cursor location may also be specified (Figure 30). These are the same parameters that may be specified (from a dialog function) when invoking the DISPLAY service.

The specified panel is fetched from the panel library and displayed as the end user would see it. Any variables referenced in the panel definition are accessed from the test pool.

Information may be entered on the panel being tested. It is stored in the corresponding variables in the test pool.

When the End PF key is pressed from the panel being tested, the option 7.1 panel is redisplayed.



```
------------------------------ TEST PANEL ---------------------------------

       THIS FUNCTION IS USED TO TEST SPF PANEL DEFINITIONS.  THE
       PANEL WILL BE DISPLAYED AS THE END USER WOULD SEE IT.  YOU MAY
       OPTIONALLY ENTER A MESSAGE ID TO BE DISPLAYED ON THE PANEL
       AND/OR THE NAME OF THE FIELD WHERE THE CURSOR IS TO BE PLACED.

ENTER THE FOLLOWING:
      PANEL NAME   ===>
      MESSAGE ID   ===>            (OPTIONAL)
      CURSOR FIELD ===>            (OPTIONAL)


ENTER THE NAME OF THE PANEL TO BE TESTED OR PRESS END KEY TO EXIT
```

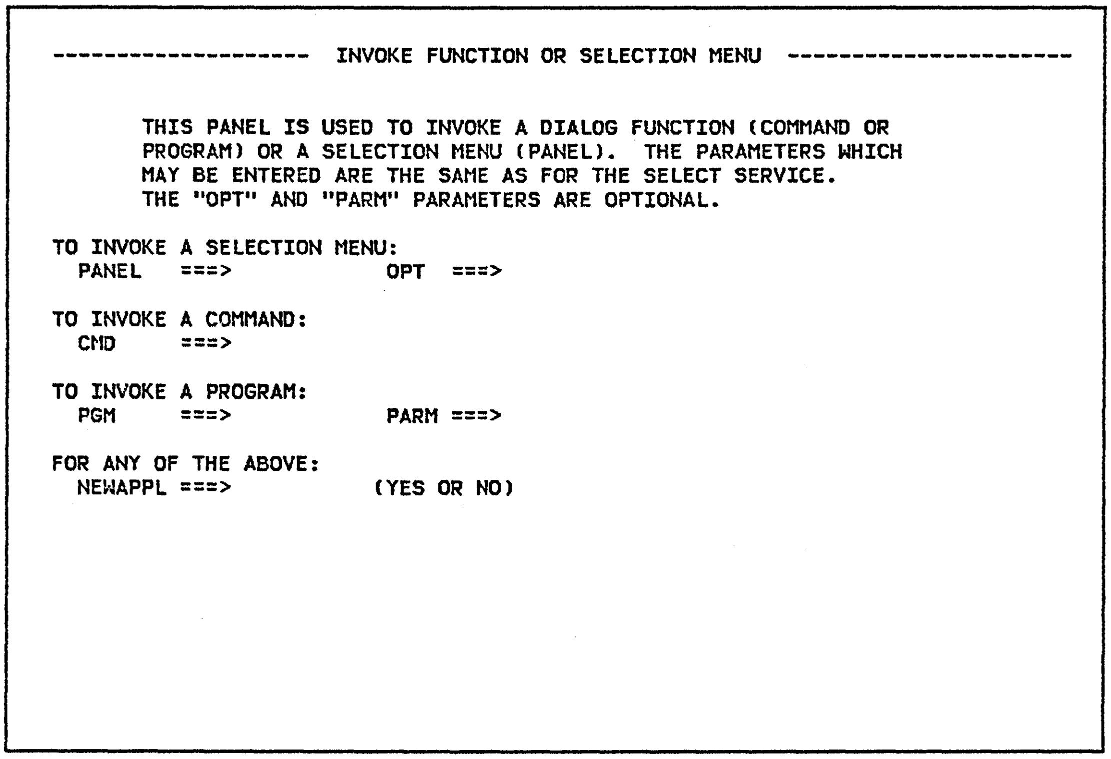
Figure 30. Entry Panel for Testing a Panel Definition

TEST FUNCTION (OPTION 7.2)

The test function option allows a dialog function or menu hierarchy to be tested without having to build "scaffolding" code.

When this option is selected, a panel is displayed that allows entry of a command or program name (to invoke a function), or a panel name to test a menu hierarchy (Figure 31). The information that may be entered on this panel corresponds to the parameters that may be specified (from a dialog function) when invoking the SELECT service.

When the invoked function completes execution, or the End PF key is pressed from the specified panel (selection menu), the test function entry panel is redisplayed.



```
-------------------- INVOKE FUNCTION OR SELECTION MENU ----------------------

       THIS PANEL IS USED TO INVOKE A DIALOG FUNCTION (COMMAND OR
       PROGRAM) OR A SELECTION MENU (PANEL).  THE PARAMETERS WHICH
       MAY BE ENTERED ARE THE SAME AS FOR THE SELECT SERVICE.
       THE "OPT" AND "PARM" PARAMETERS ARE OPTIONAL.

TO INVOKE A SELECTION MENU:
  PANEL   ===>            OPT  ===>

TO INVOKE A COMMAND:
  CMD     ===>

TO INVOKE A PROGRAM:
  PGM     ===>            PARM ===>

FOR ANY OF THE ABOVE:
  NEWAPPL ===>           (YES OR NO)
```

Figure 31. Entry Panel for Testing a Function

TEST VARIABLES (OPTION 7.3)

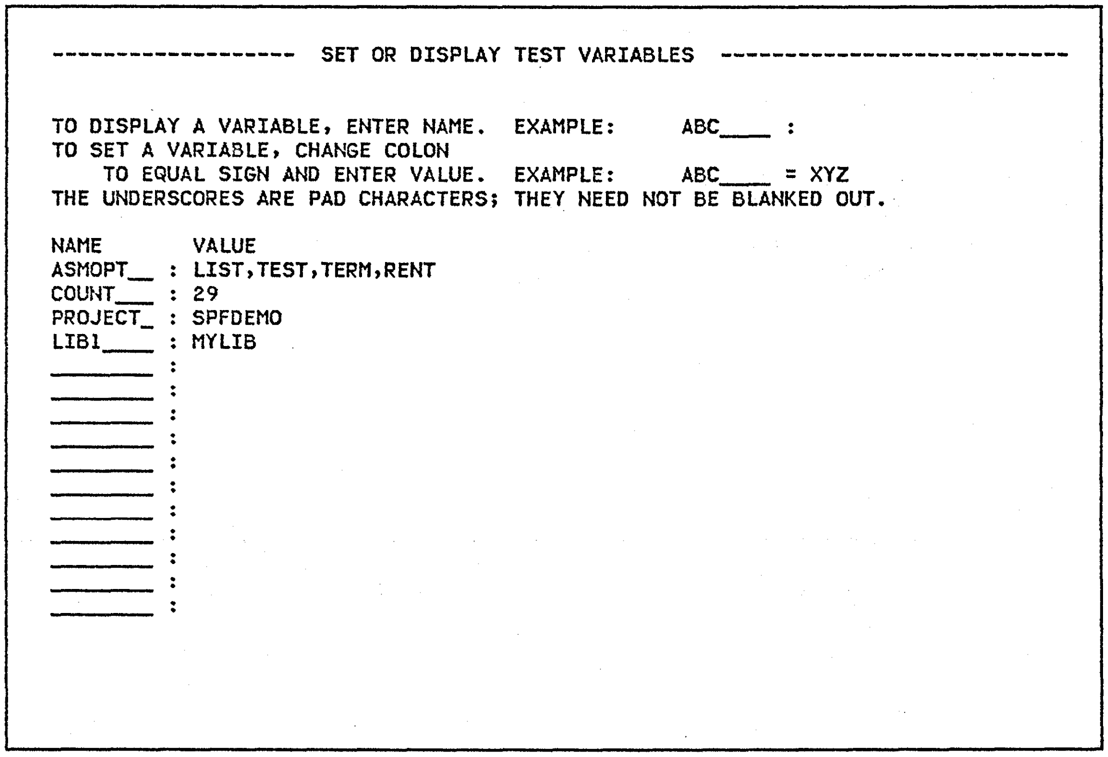
The test variables option allows dialog variables to be set and/or displayed in the test pool. It is intended for use with the test panel and test function options.

When this option is selected, a panel is displayed that allows entry of variable names down the left-hand column. This column has underscores as pad characters to indicate where the names may be entered (the underscores need not be blanked out). See Figure 32.

The current contents of a variable in the test pool may be displayed simply by entering the name of the variable and pressing the ENTER key. The value will then be displayed to the right of the colon.

The contents of a variable in the test pool may be set by entering the variable name, changing the colon to an equal sign (=), entering the desired value to the right of the equal sign, and then pressing the ENTER key.

More than one variable may be displayed or set in the same interaction.



```
------------------- SET OR DISPLAY TEST VARIABLES ---------------------------


TO DISPLAY A VARIABLE, ENTER NAME.  EXAMPLE:     ABC____ :
TO SET A VARIABLE, CHANGE COLON
    TO EQUAL SIGN AND ENTER VALUE.  EXAMPLE:     ABC____ = XYZ
THE UNDERSCORES ARE PAD CHARACTERS; THEY NEED NOT BE BLANKED OUT.

NAME        VALUE
ASMOPT__ : LIST,TEST,TERM,RENT
COUNT___ : 29
PROJECT_ : SPFDEMO
LIB1____ : MYLIB
________ :
________ :
________ :
________ :
________ :
________ :
________ :
________ :
________ :
________ :
________ :
```

Figure 32. Entry Panel for Testing a Variable

CONVERT MENUS (OPTION 7.4)

Installations that have previously extended or custom tailored SPF must ensure that the primary option menu and all lower selection menus that were displayed by the SPFUTIL program are in new format. In new SPF, these menus are displayed by the SELECT service. The SPFUTIL program no longer exists.

The convert menus option provides automated conversion of some old format menus to new format panel definitions. Two panels are displayed that are similar to the move/copy utility (option 3.3). Old format members are read from the first ("from") library, converted to the new panel format, and stored in the second ("to") library. The "from" and "to" panels are shown in Figure 33. Note that unlike the move/copy utility, there is no option selection -- it is always a copy operation.

The panels shown in Figure 33 are the MVS version of the option 7.4 panels. The VM version is similar to the move/copy utility in SPF-VM.

If a menu cannot be converted, a special panel is stored in the second library. It is a displayable "box" panel with a message indicating the name of the corresponding old menu that could not be converted.

The convert menus option will handle only two types of old format menus:

- Lower level selection menus (below the primary option level). This is limited to selection menus that were designed specifically to be processed by the SPFUTIL program in the previous SPF products.

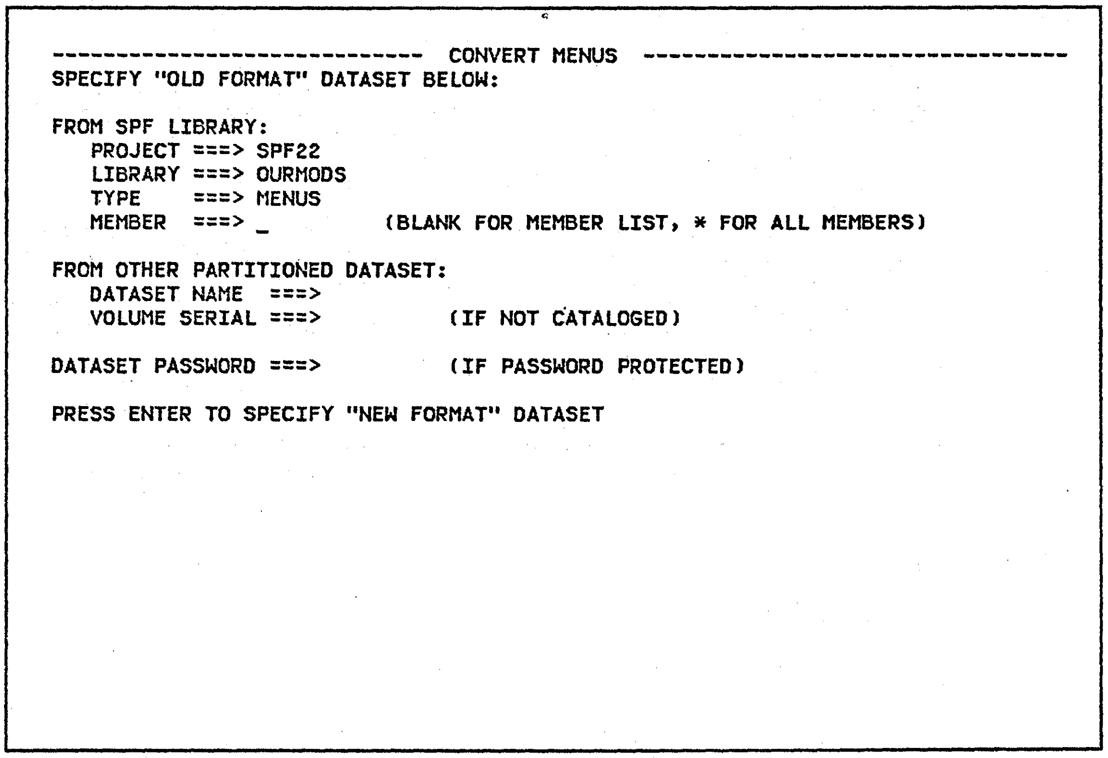
- Tutorial pages.

Do _not_ attempt to convert a primary option menu via this utility. If you have added options to the old primary option menu, you must manually add these options to the new primary option menu.

Also, no attempt should be made to convert foreground and background (batch) menus, except for the foreground selection menu (old name FORA, new name ISPFORA). The foreground selection menu must be converted to new format, and may be converted with this utility. All other foreground and background (batch) displays, including the background (batch) selection menu are supported in old format only. See _SPF Installation and Customization_ for more information.

Conversion of tutorial pages is optional; both formats are supported. If you develop additional tutorial pages, use of the new format is recommended since it is simpler than the old.

This utility cannot handle the bypassing of a tutorial page that is viewed only if explicitly selected (bypassed in the normal flow when the user keeps pressing the ENTER key). The converted page will _not_ be bypassed in the normal flow. To correct the problem, manually change the parent panel by inserting an asterisk in front of the panel name in the TRANS statement. See "Help/Tutorial Panels" in Chapter 5 for more information.



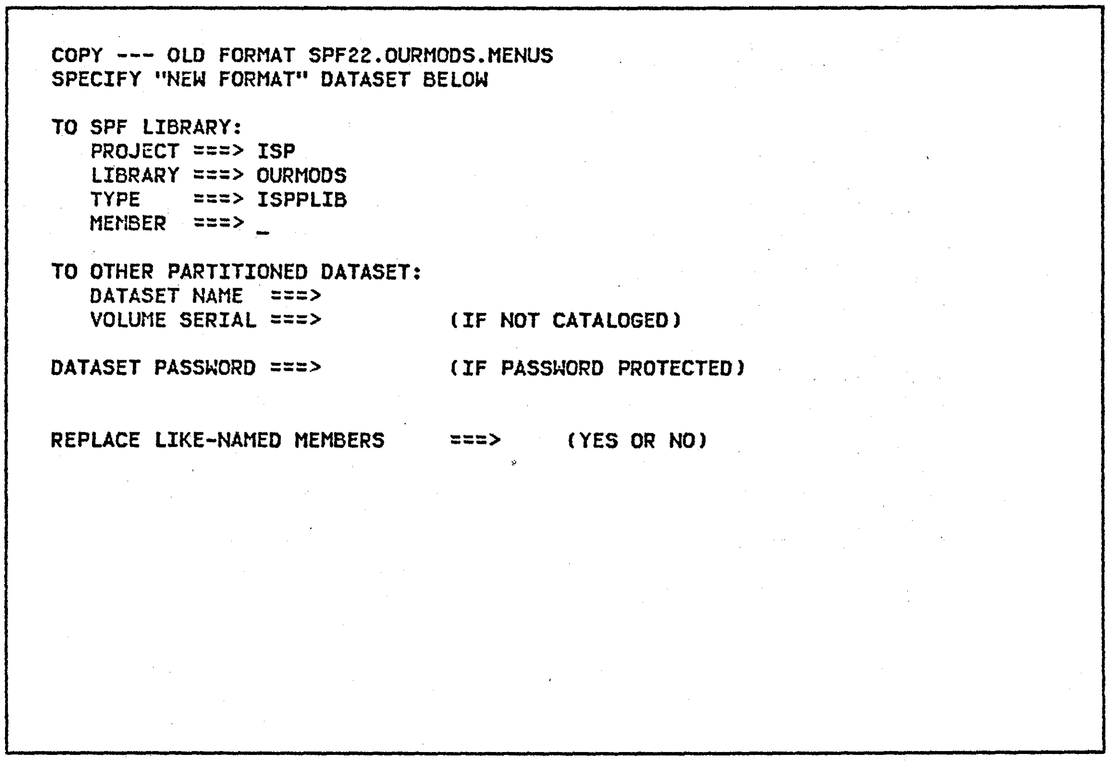
```
----------------------------- CONVERT MENUS ---------------------------------
SPECIFY "OLD FORMAT" DATASET BELOW:

FROM SPF LIBRARY:
   PROJECT ===> SPF22
   LIBRARY ===> OURMODS
   TYPE    ===> MENUS
   MEMBER  ===> _         (BLANK FOR MEMBER LIST, * FOR ALL MEMBERS)

FROM OTHER PARTITIONED DATASET:
   DATASET NAME  ===>
   VOLUME SERIAL ===>          (IF NOT CATALOGED)

DATASET PASSWORD ===>          (IF PASSWORD PROTECTED)

PRESS ENTER TO SPECIFY "NEW FORMAT" DATASET
```



```
COPY --- OLD FORMAT SPF22.OURMODS.MENUS
SPECIFY "NEW FORMAT" DATASET BELOW

TO SPF LIBRARY:
   PROJECT ===> ISP
   LIBRARY ===> OURMODS
   TYPE    ===> ISPPLIB
   MEMBER  ===> _

TO OTHER PARTITIONED DATASET:
   DATASET NAME  ===>
   VOLUME SERIAL ===>          (IF NOT CATALOGED)

DATASET PASSWORD ===>          (IF PASSWORD PROTECTED)


REPLACE LIKE-NAMED MEMBERS     ===>     (YES OR NO)
```

Figure 33. Entry Panels for Converting Menu Definitions

CONVERT MESSAGES (OPTION 7.5)

The convert messages option provides automated conversion from old format SPF message definitions to new format message definitions. As with option 7.4, two panels are displayed that are similar to the move/copy utility. Old format members are read from the first library, converted to new message format, and stored in the second library.

Variable fields in old format messages are converted to dummy variable names, beginning with an ampersand. These must be changed manually to the appropriate dialog variable names.

Generally, installations that have previously extended or custom tailored SPF should not need to convert message formats. This utility is intended to assist in development of new dialogs that use messages derived from existing (old format) messages.

The restriction on message formats is that only new format messages may be displayed on new format panels, and old format messages on old format panels. The new LOG service will write only new format messages to the SPF log file. Log messages specified via foreground and background (batch) procs must remain in old format.

TEST MENU (OPTION 7.6)

The test menu option allows old format SPF menus to be displayed as the end user would see them. Old format menus are still used for the foreground and background (batch) options in the SPF program development facility.

When this option is selected, a panel is displayed that allows entry of an old format menu name. See Figure 34. When the specified menu is displayed, the initial values for the input and variable output fields are displayed as "VAL 01", "VAL 02", etc. The numbers correspond to the numbers on the action statements in the menu definition. The menu tester supports up to 50 action statements.

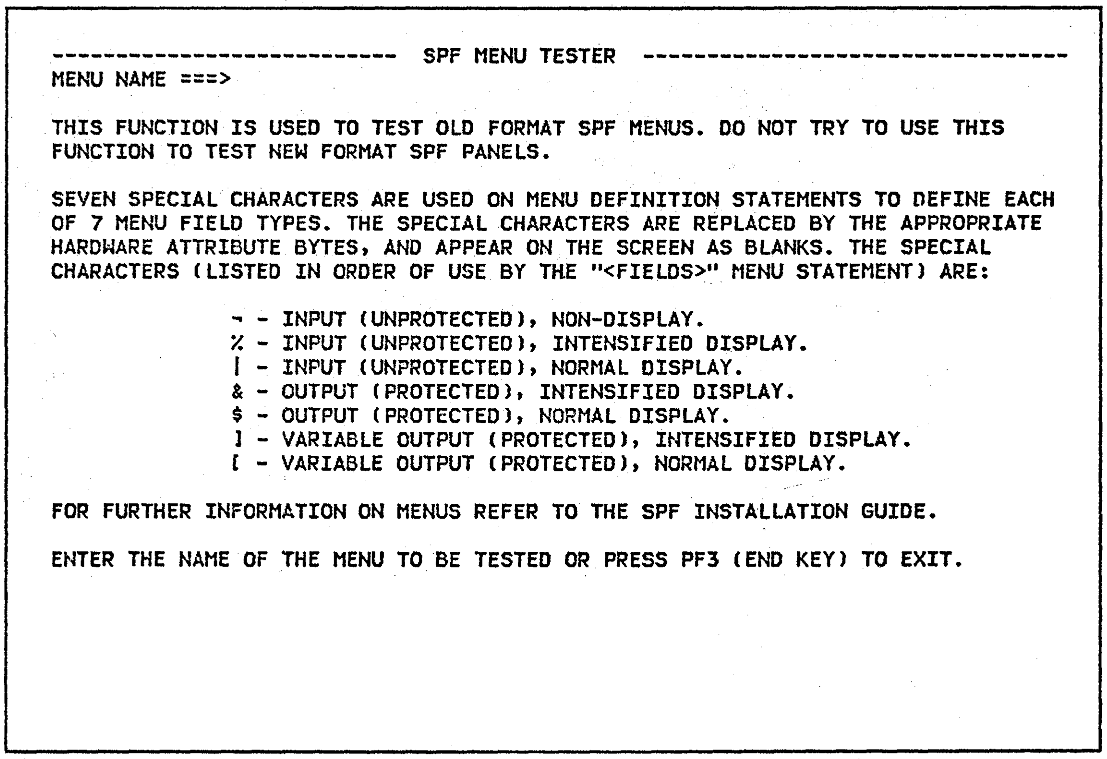
When the menu is displayed, information may be entered into the input fields. The input and variable output fields may be initialized prior to displaying the menu by first displaying a menu named SETUP. The procedure is as follows:

1. Enter SETUP as the menu name on the test menu display.

2. On the SETUP menu, fill in the desired parameters by overtyping "VAL 01", "VAL 02", etc. These values will be passed as initial values to the menu to be tested.

3. Press ENTER or the End PF key to return to the test menu display.

4. Enter the name of the menu to be tested on the test menu display.



```
---------------------------  SPF MENU TESTER  ---------------------------------
MENU NAME ===>

THIS FUNCTION IS USED TO TEST OLD FORMAT SPF MENUS. DO NOT TRY TO USE THIS
FUNCTION TO TEST NEW FORMAT SPF PANELS.

SEVEN SPECIAL CHARACTERS ARE USED ON MENU DEFINITION STATEMENTS TO DEFINE EACH
OF 7 MENU FIELD TYPES. THE SPECIAL CHARACTERS ARE REPLACED BY THE APPROPRIATE
HARDWARE ATTRIBUTE BYTES, AND APPEAR ON THE SCREEN AS BLANKS. THE SPECIAL
CHARACTERS (LISTED IN ORDER OF USE BY THE "<FIELDS>" MENU STATEMENT) ARE:

             ¬ - INPUT (UNPROTECTED), NON-DISPLAY.
             % - INPUT (UNPROTECTED), INTENSIFIED DISPLAY.
             | - INPUT (UNPROTECTED), NORMAL DISPLAY.
             & - OUTPUT (PROTECTED), INTENSIFIED DISPLAY.
             $ - OUTPUT (PROTECTED), NORMAL DISPLAY.
             ] - VARIABLE OUTPUT (PROTECTED), INTENSIFIED DISPLAY.
             [ - VARIABLE OUTPUT (PROTECTED), NORMAL DISPLAY.

FOR FURTHER INFORMATION ON MENUS REFER TO THE SPF INSTALLATION GUIDE.

ENTER THE NAME OF THE MENU TO BE TESTED OR PRESS PF3 (END KEY) TO EXIT.
```

Figure 34. Panel for Testing Old Format Menus

This appendix illustrates the implementation of an "employee records" application. The dialog begins with the display of a primary option menu, from which the user may select several options. Only the first option is implemented in this example.

The overall organization of the dialog is shown in Figure 35. The panel definition for the primary option menu, named EMPL, is shown in Figure 36. For the first option, the menu invokes a dialog function coded as a command procedure named EMPLCMD.

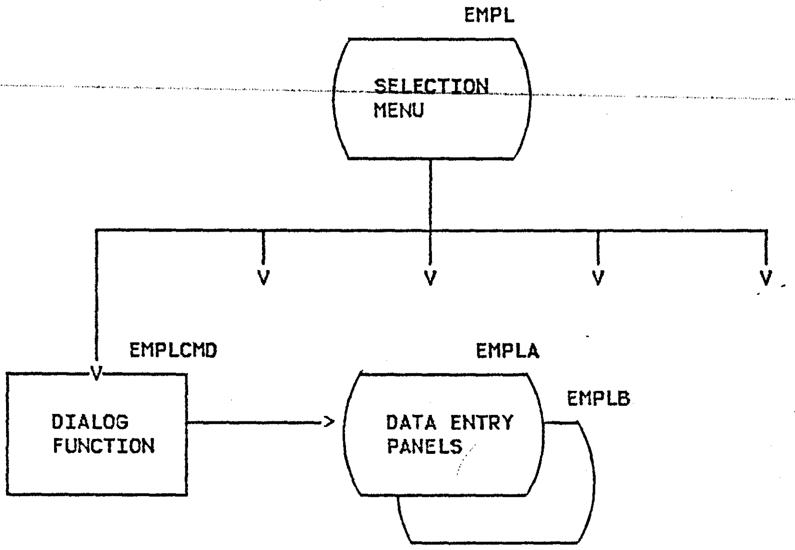


Figure 35. Sample Problem - Overall Dialog Organization

```
%----------------------------  EMPLOYEE RECORDS  ------------------------------
%SELECT OPTION ===>_OPT      %
%
%   1 +MODIFY      - ADD, UPDATE, OR DELETE EMPLOYEE RECORDS
%   2 +(FUTURE)    - FUNCTION NOT YET AVAILABLE
%   3 +(FUTURE)    - FUNCTION NOT YET AVAILABLE
%   4 +(FUTURE)    - FUNCTION NOT YET AVAILABLE
%   5 +(FUTURE)    - FUNCTION NOT YET AVAILABLE
%
+PRESS%END KEY+TO TERMINATE

)PROC
  &SEL = TRANS( TRUNC (&OPT,'.')
                1,'CMD(EMPLCMD)'

          /* FUTURE OPTIONS TO GO HERE */

               ' ',' '
                *,'?' )
)END
```

Figure 36. Sample Problem - Primary Option Menu (EMPL)

Function EMPLCMD displays two data entry panels, named EMPLA and EMPLB. The first panel prompts the user to enter an employee serial number. The second panel allows the user to add, update, or delete records from an employee table. The table contains the serial number, name, home address, and phone number for some group of employees.

The panel definitions for the two data entry panels are shown in Figure 37 and Figure 38.

VER statements in the first panel definition verify that an employee serial number was entered, and that it consists of 6 numeric digits.

VER statements in the second panel definition verify that the user did not make an invalid change to the "type of change" field; i.e. did not specify NEW if the employee record already exists, nor UPDATE or DELETE if the record does not exist. (See description of the EMPLCMD procedure.) Additional VER statements check the validity of the user's inputs and ensure that all required fields have been entered for a type change of NEW or UPDATE.

The messages that may be displayed on these panels are shown in Figure 39. (Message id EMPX216 is not used in this sample problem.)

```
%---------------------------- EMPLOYEE SERIAL ------------------------------
%ENTER EMPLOYEE SERIAL BELOW

+   EMPLOYEE SERIAL%===>_EMPSER+     (MUST BE 6 NUMERIC DIGITS)


+PRESS%ENTER+TO DISPLAY EMPLOYEE RECORD.
+PRESS%END+KEY TO RETURN TO PREVIOUS MENU.

)PROC
  VER (&EMPSER,NONBLANK)
  VER (&EMPSER,NUM)

)END
```

Figure 37. Sample Problem - First Data Entry Panel (EMPLA)

```
%----------------------------  EMPLOYEE RECORDS  ------------------------------
%EMPLOYEE SERIAL: &EMPSER

+   TYPE OF CHANGE%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)

+   EMPLOYEE NAME:
+     LAST    %===>_LNAME         +
+     FIRST   %===>_FNAME         +
+     INITIAL%===>_I+

+   HOME ADDRESS:
+     LINE 1 %===>_ADDR1                                      +
+     LINE 2 %===>_ADDR2                                      +
+     LINE 3 %===>_ADDR3                                      +
+     LINE 4 %===>_ADDR4                                      +

+   HOME PHONE:
+     AREA CODE    %===>_PHA+
+     LOCAL NUMBER%===>_PHNUM    +

)INIT
  .HELP = PERS032
  .CURSOR = TYPECHG
  IF (&PHA = ' ')
    &PHA = 301
  &TYPECHG = TRANS (&TYPECHG N,NEW U,UPDATE D,DELETE)

)PROC
  &TYPECHG = TRUNC (&TYPECHG,1)
  VER (&TYPECHG,NONBLANK)
  VER (&TYPECHG,LIST,N,U,D,MSG=EMPX210)
  IF (&TYPECHG = N)
    IF (&CHKTYPE ¬= N)
      .MSG = EMPX211
  IF (&TYPECHG ¬= N)
    IF (&CHKTYPE = N)
      .MSG = EMPX212
  VER (&LNAME,ALPHA)
  VER (&FNAME,ALPHA)
  VER (&I,ALPHA)
  VER (&PHA,NUM)
  VER (&PHNUM,PICT,'NNN-NNNN')
  IF (&TYPECHG = N,U)
    VER (&LNAME,NONBLANK,MSG=EMPX214)
    VER (&FNAME,NONBLANK,MSG=EMPX213)
    VER (&ADDR1,NONBLANK,MSG=EMPX215)
    VER (&ADDR2,NONBLANK,MSG=EMPX215)
    VER (&ADDR3,NONBLANK,MSG=EMPX215)

)END
```

Figure 38. Sample Problem - Second Data Entry Panel (EMPLB)

```
EMPX210  'INVALID TYPE OF CHANGE'    .HELP=PERS033   .ALARM=YES
'TYPE OF CHANGE MUST BE NEW, UPDATE, OR DELETE.'

EMPX211  'TYPE ''NEW'' INVALID'      .HELP=PERS033   .ALARM=YES
'EMPLOYEE SERIAL &EMPSER ALREADY EXISTS.  CANNOT BE SPECIFIED AS NEW.'

EMPX212  'UPDATE OR DELETE INVALID'  .HELP=PERS033   .ALARM=YES
'EMPLOYEE SERIAL &EMPSER IS NEW.  CANNOT SPECIFY UPDATE OR DELETE.'

EMPX213  'ENTER FIRST NAME'          .HELP=PERS034   .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE = NEW OR UPDATE.'

EMPX214  'ENTER LAST NAME'           .HELP=PERS034   .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE = NEW OR UPDATE.'

EMPX215  'ENTER HOME ADDRESS'        .HELP=PERS035   .ALARM=YES
'HOME ADDRESS MUST BE ENTERED FOR TYPE OF CHANGE = NEW OR UPDATE.'

EMPX216  'AREA CODE INVALID'         .ALARM=YES
'AREA CODE &PHA IS NOT DEFINED.  PLEASE CHECK THE PHONE BOOK.'

EMPX217  '&EMPSER ADDED'
'EMPLOYEE &LNAME, &FNAME &I ADDED TO FILE.'

EMPX218  '&EMPSER UPDATED'
'RECORDS FOR &LNAME, &FNAME &I UPDATED.'

EMPX219  '&EMPSER DELETED'
'RECORDS FOR &LNAME, &FNAME &I DELETED.'
```

Figure 39. Sample Problem - Messages (Member EMPX)

The code for function EMPLCMD, coded as a CLIST, is shown in Figure 40. The same function coded as a PL/I program named EMPLPGM is shown in Figure 41 through Figure 43. If the PL/I program were to receive control from the primary option menu (EMPL), the selection keywords on that menu would have to be changed

```
    from:  1,'CMD(EMPLCMD)'
    to:    1,'PGM(EMPLPGM)'
```

The function starts by invoking the TBOPEN service to open the employee table, named EMPLTBL. If the table does not exist (first execution of the function), the function invokes TBCREATE to create it. The function then displays the first data entry panel (EMPLA) via invoking the DISPLAY service.

The employee serial, entered by the user on the first panel, serves as the key variable for accessing the employee table. After the first panel has been displayed, the function attempts to read the row from the table for this employee, using the TBGET service. Then it initializes variables for the second data entry panel (EMPLB) and displays the panel.

If the employee record was found in the table, the "type of change" field on panel EMPLB is initialized to UPDATE (the user may change it to DELETE). The other fields in the panel are automatically initialized to the values read from the table. If the employee record was not found, the "type of change" field is initialized to NEW, and the other input fields on the panel are initialized to blanks.

Once the user has correctly entered all required infomation on the second panel and pressed the ENTER key, the function updates the employee table, using the TBADD, TBPUT, or TBDELETE service. It then redisplays the first panel (EMPLA) with a confirmation message. It also writes the confirmation to the log file, using the LOG service.

If the user presses the End PF key from the second panel, the first panel is redisplayed without making any changes to the table, and without displaying or logging a message.

When the first panel is redisplayed, the user may enter another employee serial number, and the process repeats. If the user presses the End PF key from the first panel, the function closes the employee table using the TBCLOSE service and completes execution. The selection menu from which it was invoked will then be redisplayed.

```
PROC 0                                /* EMPLOYEE UPDATE FUNCTION    */
  CONTROL MAIN                              /*                         . */
  SET &STATE   = 1                          /*INITIAL ENTRY STATE    */
  SET &EMPSER  =                            /*INITIALIZE EMPL SERIAL*/
  SET &MSG   =                              /*INITIALIZE MESSAGE     */
  ISPEXEC TBOPEN EMPLTBL                    /*OPEN EMPLOYEE TABLE    */
  IF &LASTCC ¬=0 THEN                       /*IF TABLE DOESN'T EXIST*/-
    ISPEXEC TBCREATE EMPLTBL KEYS(EMPSER)   /*CREATE IT              */-
            NAMES(LNAME FNAME I ADDR1 ADDR2 ADDR3 ADDR4 PHA PHNUM)
  DO WHILE &STATE ¬= 4                      /*LOOP UNTIL TERM SET    */
    ISPEXEC DISPLAY PANEL(EMPLA) MSG(&MSG)  /*SELECT EMPLOYEE        */
    IF &LASTCC=8 THEN SET &STATE=4          /*END KEY PRESSED        */
    ELSE DO                                 /*ENTER KEY PRESSED      */
      SET &MSG =                            /*RESET MESSAGE          */
      SET &STATE = 2                        /*PROCESS EMPLOYEE PANEL*/
      ISPEXEC TBGET EMPLTBL                 /*OBTAIN EMPLOYEE DATA  */
      IF &LASTCC= 0 THEN SET &TYPECHG = U   /*RECORD EXISTS-UPDATE  */
      ELSE DO                               /*RECORD DOES NOT EXIST */
        SET &TYPECHG = N                    /*SET TYPE = NEW         */
        SET &LNAME   =                      /*                       */
        SET &FNAME   =                      /*  INITIALIZE           */
        SET &I       =                      /*    PANEL              */
        SET &ADDR1   =                      /*      VARIABLES        */
        SET &ADDR2   =                      /*        TO NULL        */
        SET &ADDR3   =                      /*                       */
        SET &ADDR4   =                      /*                       */
        SET &PHA     =                      /*                       */
        SET &PHNUM   =                      /*                       */
        END                                 /*                       */
      SET &CHKTYPE = &TYPECHG               /*SAVE TYPE OF CHANGE    */
      ISPEXEC DISPLAY PANEL(EMPLB)          /*DISPLAY EMPLOYEE DATA */
      IF &LASTCC ¬= 8 THEN DO               /*END KEY NOT PRESSED    */
        IF &TYPECHG = N  THEN DO            /*IF NEW EMPLOYEE        */
          ISPEXEC TBADD EMPLTBL             /*ADD EMPLOYEE TO TABLE */
          SET &MSG = EMPX217                /*EMPLOYEE ADDED MESSAGE*/
          END                               /*                       */
        ELSE DO                             /*                       */
          IF &TYPECHG = U  THEN DO          /*IF UPDATE REQUESTED    */
            ISPEXEC TBPUT EMPLTBL           /*  UPDATE TABLE         */
            SET &MSG = EMPX218              /*  UPDATE MESSAGE       */
            END                             /*                       */
          ELSE DO                           /*                       */
            ISPEXEC TBDELETE EMPLTBL        /*DELETE TBL MEMBER      */
            SET &MSG = EMPX219              /*EMPLOYEE DELETED MSG  */
            END                             /*                       */
          END                               /*END TABLE MODS         */
        END                                 /*END 2ND PANEL PROCESS */
      END                                   /*END 1ST PANEL PROCESS */
    IF &MSG ¬=  THEN ISPEXEC LOG MSG(&MSG)  /*LOG ANY MESSAGES       */
  END                                       /*END DO LOOP            */
  ISPEXEC TBCLOSE EMPLTBL                   /*CLOSE TABLE            */
EXIT CODE(0)
END
```

Figure 40. Sample Problem - CLIST (EMPLCMD)

```
EMPLPGM: PROC OPTIONS(MAIN);           /*EMPLOYEE UPDATE FUNCTION */
 %INCLUDE EMPLDCL;                         /*DCL & DEFINE VARIABLES*/
 MSG  = ' ';                               /*INITIALIZE MESSAGE    */
 CALL ISPLINK('TBOPEN', 'EMPLTBL ');       /*OPEN TABLE            */
 IF PLIRETV() ¬= 0 THEN                    /*IF TABLE DOESN'T EXIST*/
   CALL ISPLINK('TBCREATE', 'EMPLTBL ', '(EMPSER)',   /*CREATE IT  */
        '(LNAME FNAME I ADDR1 ADDR2 ADDR3 ADDR4 PHA PHNUM)' );
 DO WHILE (STATE ¬= '4');                  /*LOOP UNTIL TERM SET   */
   CALL ISPLINK('DISPLAY', 'EMPLA ', MSG);       /*SELECT EMPLOYEE */
   IF PLIRETV() = 8 THEN                   /*IF END KEY PRESSED    */
     STATE = '4';                          /*  TERMINATE           */
   ELSE DO;                                /*ENTER KEY PRESSED     */
     MSG = ' ' ;                           /*RESET MESSAGE         */
     STATE = '2';                          /*PROCESS EMPLOYEE PANEL*/
     CALL ISPLINK('TBGET', 'EMPLTBL ');    /*OBTAIN EMPLOYEE DATA  */
     IF PLIRETV() = 0 THEN                 /*IF RECORD EXISTS      */
       TYPECHG ='U';                       /*  SET UPDATE FLAG     */
     ELSE DO;                              /*RECORD DOES NOT EXIST */
       TYPECHG = 'N' ;                     /*   SET TYPE = NEW     */
       LNAME     = ' ' ;                   /*                      */
       FNAME     = ' ' ;                   /*  INITIALIZE          */
       I         = ' ' ;                   /*    PANEL             */
       ADDR1     = ' ' ;                   /*      VARIABLES       */
       ADDR2     = ' ' ;                   /*        TOO NULL      */
       ADDR3     = ' ' ;                   /*                      */
       ADDR4     = ' ' ;                   /*                      */
       PHA       = ' ' ;                   /*                      */
       PHNUM     = ' ' ;                   /*                      */
       END;                                /*                      */
     CHKTYPE = TYPECHG;                    /*SAVE TYPE OF CHANGE   */
     CALL ISPLINK('DISPLAY', 'EMPLB ');    /*DISPLAY EMPLOYEE DATA */
     IF PLIRETV() ¬= 8 THEN DO;            /*END KEY NOT PRESSED   */
       IF TYPECHG = 'N' THEN DO;           /*IF NEW EMPLOYEE       */
         CALL ISPLINK('TBADD', 'EMPLTBL ');  /* ADD TO TABLE       */
         MSG = 'EMPX217 ';                 /*EMPLOYEE ADDED MESSAGE*/
         END;                              /*                      */
       ELSE DO;                            /*                      */
         IF TYPECHG = 'U' THEN DO;         /*IF UPDATE REQUESTED   */
           CALL ISPLINK('TBPUT', 'EMPLTBL ');  /* UPDATE TABLE     */
           MSG = 'EMPX218 ';               /*  UPDATE MESSAGE      */
           END;                            /*                      */
         ELSE DO;                          /*ELSE ASSUME DELETE    */
           CALL ISPLINK('TBDELETE', 'EMPLTBL ');
           MSG = 'EMPX219 ' ;              /*EMPLOYEE DELETED MSG  */
           END;                            /*                      */
         END;                              /*END TABLE MODS        */
       END;                                /*END 2ND PANEL PROCESS */
     END;                                  /*END 1ST PANEL PROCESS */
   IF MSG ¬= ' ' THEN  CALL ISPLINK('LOG', MSG);    /*LOG MSG      */
 END;                                     /*END DO LOOP            */
 CALL ISPLINK('TBCLOSE', 'EMPLTBL ');     /*CLOSE TABLE            */
%INCLUDE EMPLDEL;                         /*DELETE DEFINED VARS    */
RETURN(0);
END EMPLPGM;
```

Figure 41. Sample Problem - PL/I Main Program (EMPLPGM)

```
/*                                                                     */
/* DECLARE STATEMENTS AND VARIABLE DEFINITIONS FOR "EMPLPGM"           */
/*                                                                     */
DCL ISPLINK EXTERNAL ENTRY  OPTIONS(ASM RETCODE) ;
DCL PLIRETV BUILTIN   ;
DCL LENGTH  BUILTIN   ;
DCL RC FIXED BIN(31,0) INIT(0);
DCL EMPSER CHAR(6) INIT((6)' ');
DCL FNAME CHAR(16) INIT((16)' ');
DCL LNAME CHAR(16) INIT((16)' ');
DCL I     CHAR(1)  INIT(' ');
DCL ADDR1 CHAR(40) INIT((40)' ');
DCL ADDR2 CHAR(40) INIT((40)' ');
DCL ADDR3 CHAR(40) INIT((40)' ');
DCL ADDR4 CHAR(40) INIT((40)' ');
DCL PHA   CHAR(3) INIT((3)' ');
DCL PHNUM CHAR(8) INIT((8)' ');
DCL MSG   CHAR(8) INIT((8)' ');
DCL TYPECHG CHAR(1) INIT(' ');
DCL CHKTYPE CHAR(1) INIT(' ');
DCL STATE CHAR(1) INIT('1');            /*INITIAL ENTRY STATE      */
/*                                                                     */
/* LENGTH PARAMETER IN 'CALL ISPLINK VDEFINE' MUST BE FULL WORD.       */
/*                                                                     */
DCL LEMPSER FIXED BIN(31,0) ;
DCL LFNAME  FIXED BIN(31,0) ;
DCL LLNAME  FIXED BIN(31,0) ;
DCL LI      FIXED BIN(31,0) ;
DCL LADDR1  FIXED BIN(31,0) ;
DCL LADDR2  FIXED BIN(31,0) ;
DCL LADDR3  FIXED BIN(31,0) ;
DCL LADDR4  FIXED BIN(31,0) ;
DCL LPHA    FIXED BIN(31,0) ;
DCL LPHNUM  FIXED BIN(31,0) ;
DCL LTYPECH FIXED BIN(31,0) ;
DCL LCHKTYP FIXED BIN(31,0) ;
LEMPSER = LENGTH(EMPSER)    ;
LFNAME  = LENGTH(FNAME)     ;
LLNAME  = LENGTH(LNAME)     ;
LI      = LENGTH(I)         ;
LADDR1  = LENGTH(ADDR1)     ;
LADDR2  = LENGTH(ADDR2)     ;
LADDR3  = LENGTH(ADDR3)     ;
LADDR4  = LENGTH(ADDR4)     ;
LPHA    = LENGTH(PHA)       ;
LPHNUM  = LENGTH(PHNUM)     ;
LTYPECH = LENGTH(TYPECHG)   ;
LCHKTYP = LENGTH(CHKTYPE)   ;
/*                                                                     */
/*DEFINE VARIABLES FOR DIALOG SERVICE USE                              */
/*                                                                     */
CALL ISPLINK('VDEFINE','(EMPSER)',EMPSER,'CHAR',LEMPSER) ;
CALL ISPLINK('VDEFINE','(FNAME)',FNAME,'CHAR',LFNAME) ;
CALL ISPLINK('VDEFINE','(LNAME)',LNAME,'CHAR',LLNAME) ;
CALL ISPLINK('VDEFINE','(I)',I,'CHAR',LI) ;
CALL ISPLINK('VDEFINE','(ADDR1)',ADDR1,'CHAR',LADDR1) ;
CALL ISPLINK('VDEFINE','(ADDR2)',ADDR2,'CHAR',LADDR2) ;
CALL ISPLINK('VDEFINE','(ADDR3)',ADDR3,'CHAR',LADDR3) ;
CALL ISPLINK('VDEFINE','(ADDR4)',ADDR4,'CHAR',LADDR4) ;
CALL ISPLINK('VDEFINE','(PHA)',PHA,'CHAR',LPHA) ;
CALL ISPLINK('VDEFINE','(PHNUM)',PHNUM,'CHAR',LPHNUM) ;
CALL ISPLINK('VDEFINE','(TYPECHG)',TYPECHG,'CHAR',LTYPECH);
CALL ISPLINK('VDEFINE','(CHKTYPE)',CHKTYPE,'CHAR',LCHKTYP);
/*                                                                     */
```

Figure 42. Sample Problem - PL/I Included Segment (EMPLDCL)

```
/*                                                                    */
/*         DELETE VARIABLE DEFINITIONS FOR "EMPLPGM"                  */
/*                                                                    */
CALL ISPLINK('VDELETE','(EMPSER)');
CALL ISPLINK('VDELETE','(FNAME)') ;
CALL ISPLINK('VDELETE','(LNAME)') ;
CALL ISPLINK('VDELETE','(I)')     ;
CALL ISPLINK('VDELETE','(ADDR1)') ;
CALL ISPLINK('VDELETE','(ADDR2)') ;
CALL ISPLINK('VDELETE','(ADDR3)') ;
CALL ISPLINK('VDELETE','(ADDR4)') ;
CALL ISPLINK('VDELETE','(PHA)')   ;
CALL ISPLINK('VDELETE','(PHNUM)') ;
CALL ISPLINK('VDELETE','(TYPECHG)');
CALL ISPLINK('VDELETE','(CHKTYPE)');
/*                                                                    */
```

Figure 43. Sample Problem - PL/I Included Segment (EMPLDEL)

This appendix contains a quick reference summary of dialog services. The command invocation syntax for all services is shown first, followed by the call invocation syntax.

COMMAND INVOCATION SYNTAX

Display Services

```
ISPEXEC  DISPLAY   [PANEL(panel-name)]
                   [MSG(msg-id)]
                   [CURSOR(field-name)]

ISPEXEC  TBDISPL   table-name  PANEL(panel-name)
                               [MSG(msg-id)]
```

Table Services - General

```
ISPEXEC  TBCREATE  table-name  [KEYS(key-name-list)]
                               [NAMES(name-list)]
                               [WRITE/NOWRITE]
                               [REPLACE]

ISPEXEC  TBOPEN    table-name  [WRITE/NOWRITE]

ISPEXEC  TBQUERY   table-name  [KEYS(key-name)]
                               [NAMES(var-name)]
                               [ROWNUM(rownum-name)]
                               [KEYNUM(keynum-name)]
                               [NAMENUM(namenum-name)]
                               [POSITION(crp-name)]

ISPEXEC  TBSAVE    table-name  [NEWCOPY/REPLCOPY]
                               [NAME(alt-name)]
                               [PAD(percentage)]

ISPEXEC  TBCLOSE   table-name  [NEWCOPY/REPLCOPY]
                               [NAME(alt-name)]
                               [PAD(percentage)]
ISPEXEC  TBEND     table-name

ISPEXEC  TBERASE   table-name
```

Table Services - Row Operations

```
ISPEXEC  TBADD     table-name  [SAVE(name-list)]

ISPEXEC  TBDELETE  table-name

ISPEXEC  TBGET     table-name  [SAVENAME(var-name)]

ISPEXEC  TBPUT     table-name  [SAVE(name-list)]

ISPEXEC  TBMOD     table-name  [SAVE(name-list)]

ISPEXEC  TBEXIST   table-name

ISPEXEC  TBSCAN    table-name  [ARGLIST(name-list)]
                               [SAVENAME(var-name)]
```

```
ISPEXEC  TBSARG    table-name  [ARGLIST(name-list)]

ISPEXEC  TBTOP     table-name

ISPEXEC  TBBOTTOM  table-name  [SAVENAME(var-name)]

ISPEXEC  TBSKIP    table-name  [NUMBER(number)]
                               [SAVENAME(var-name)]

ISPEXEC  TBVCLEAR  table-name
```

File Tailoring Services

```
ISPEXEC  FTOPEN    [TEMP]

ISPEXEC  FTINCL    skel-name  [NOFT]

ISPEXEC  FTCLOSE   [NAME(member-name)]

ISPEXEC  FTERASE   member-name
```

Variable Services

```
ISPEXEC  VGET  name-list  [SHARED/PROFILE]

ISPEXEC  VPUT  name-list  [SHARED/PROFILE]
```

Other Services

```
|  ISPEXEC  SELECT  { PANEL(panel-name)  [OPT(option)]      }
                    { CMD(command)                          }
                    { PGM(program-name)  [PARM(parameters)] }
                    [NEWAPPL/NEWPOOL]

   ISPEXEC  CONTROL { DISPLAY  { LINE [START(line-number)] } }
                    {          { SM   [START(line-number)] } }
                    {          { REFRESH                   } }
                    { NONDISPL [ENTER/END]                   }
                    { ERRORS   [CANCEL/RETURN]               }
|                   { SPLIT    { ENABLE  }                   }
|                   {          { DISABLE }                   }

   ISPEXEC  BROWSE  DATASET(dsname)   [VOLUME(serial)]
                                      [PASSWORD(pswd-value)]

   ISPEXEC  BROWSE  FILE(fileid)      [MEMBER(member-name)]

   ISPEXEC  EDIT    DATASET(dsname)   [VOLUME(serial)]
                                      [PASSWORD(pswd-value)]

   ISPEXEC  EDIT    FILE(fileid)      [MEMBER(member-name)]

   ISPEXEC  LOG     MSG(msg-id)
```

CALL INVOCATION SYNTAX

Display Services

```
CALL   ISPLINK ('DISPLAY'   [,panel-name]
                            [,msg-id]
                            [,field-name] );

CALL   ISPLINK ('TBDISPL',  table-name, panel-name
                                        [,msg-id] );
```

Table Services - General

```
CALL   ISPLINK ('TBCREATE', table-name, key-name-list
                                        [,name-list]
                                        [,'WRITE/NOWRITE']
                                        [,'REPLACE'] );

CALL   ISPLINK ('TBOPEN',   table-name  [,'WRITE'/'NOWRITE'] );

CALL   ISPLINK ('TBQUERY',  table-name  [,key-name]
                                        [,var-name]
                                        [,rownum-name]
                                        [,keynum-name]
                                        [,namenum-name]
                                        [,crp-name] );

CALL   ISPLINK ('TBSAVE',   table-name, ['NEWCOPY/REPLCOPY']
                                        [,alt-name]
                                        [,percentage] );

CALL   ISPLINK ('TBCLOSE',  table-name  [,'NEWCOPY/REPLCOPY']
                                        [,alt-name]
                                        [,percentage] );

CALL   ISPLINK ('TBEND',    table-name);

CALL   ISPLINK ('TBERASE',  table-name);
```

Table Services - Row Operations

```
CALL   ISPLINK ('TBADD',    table-name [,name-list] );

CALL   ISPLINK ('TBDELETE', table-name);

CALL   ISPLINK ('TBGET',    table-name [,var-name] );

CALL   ISPLINK ('TBPUT',    table-name [,name-list] );

CALL   ISPLINK ('TBMOD',    table-name [,name-list] );

CALL   ISPLINK ('TBEXIST',  table-name);

CALL   ISPLINK ('TBSCAN',   table-name [,name-list]
                                       [,var-name] );

CALL   ISPLINK ('TBSARG',   table-name [,name-list] );

CALL   ISPLINK ('TBTOP',    table-name);

CALL   ISPLINK ('TBBOTTOM', table-name [,var-name] );

CALL   ISPLINK ('TBSKIP',   table-name [,number]
                                       [,var-name] );

CALL   ISPLINK ('TBVCLEAR', table-name);
```

File Tailoring Services

```
CALL  ISPLINK ('FTOPEN'    [,'TEMP'] );
CALL  ISPLINK ('FTINCL',   skel-name [,'NOFT'] );
CALL  ISPLINK ('FTCLOSE'   [,member-name] );
CALL  ISPLINK ('FTERASE',  member-name);
```

Variable Services

```
CALL  ISPLINK ('VGET',     name-list [,'SHARED'/'PROFILE'] );
CALL  ISPLINK ('VPUT',     name-list [,'SHARED'/'PROFILE'] );
CALL  ISPLINK ('VDEFINE',  name-list, variable, format, length);
CALL  ISPLINK ('VDELETE',  name-list);
CALL  ISPLINK ('VCOPY',    var-name, length, variable
                           [,'LOCATE'/'MOVE'] );
CALL  ISPLINK ('VREPLACE', var-name, length, value);
CALL  ISPLINK ('VRESET');
```

Other Services

```
CALL  ISPLINK ('SELECT',  buf-length, buffer);
CALL  ISPLINK ('CONTROL', type [,mode]
                               [,line-number] );
CALL  ISPLINK ('BROWSE',  dsname [,serial]
                                 [,pswd-value] );
CALL  ISPLINK ('BROWSE',  fileid [,member-name] );
CALL  ISPLINK ('EDIT',    dsname [,serial]
                                 [,pswd-value] );
CALL  ISPLINK ('EDIT',    fileid [,member-name] );
CALL  ISPLINK ('LOG',     msg-id);
```

System Productivity Facility
Dialog Management Services SC34-2036-1

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. It will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comments are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name and mailing address:

____

____

____

What is your occupation? ____________________

Number of latest Technical Newsletter (if any) concerning this publication: __________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

Cut or Fold Along Line

SC34-2036-1

Reader's Comment Form

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department Z59, Building 931
P. O. Box 390
Poughkeepsie, New York 12602

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

Cut or Fold Along Line

SPF: Dialog Management Services (File No. S370/4300-39) Printed in U.S.A. SC34-2036-1

System Productivity Facility
Dialog Management Services SC34-2036-1

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. It will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comments are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name and mailing address:

____

____

____

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

Cut or Fold Along Line

What is your occupation? ______________________________

Number of latest Technical Newsletter (if any) concerning this publication: _______________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

SC34-2036-1

Reader's Comment Form

Fold and tape — **Please Do Not Staple** — Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department Z59, Building 931
P. O. Box 390
Poughkeepsie, New York 12602

Fold and tape — **Please Do Not Staple** — Fold and tape

IBM

**International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604**

**IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591**

**IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601**

Cut or Fold Along Line

SPF: Dialog Management Services (File No. S370/4300-39) Printed in U.S.A. SC34-2036-1