



---

Interactive  
System Productivity Facility  
(ISPF)

---

Dialog Management  
Services

---

---

MVS, VM/SP, and VSE/AF

---

---

Publication Number  
SC34-2088-2

File Number  
S370/4300-39

---

Program Number  
5668-960

Third Edition (March 1985)

This edition applies to the program product Interactive System Productivity Facility (ISPF) as shown in the table below, and to all subsequent releases and modifications until otherwise indicated in new editions or Technical Newsletters.

System Release	MVS 3.8	VM/SP 1.0	VSE/AF 1.3.5 5746-XE8	VSE/AF 2.1 5666-301
ISPF	5668-960	5668-960	5668-960	5668-960

Changes are made periodically to the information herein. Therefore, before using this publication, consult the latest IBM System 370 and 4300 Processors Bibliography (GC20-0001) for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this document is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the address given below. Requests for copies of IBM publications should be made to your IBM representative or to the branch office serving your locality.

A form for readers' comments has been provided at the back of this publication. If this form has been removed, address comments to: IBM Corporation, Systems Publications, Dept. T46, P.O. Box 60000, Cary, NC 27511. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

## PREFACE

The Interactive System Productivity Facility (ISPF) and the ISPF/Program Development Facility (ISPF/PDF or PDF) are related IBM program products. Together, they are designed to improve user productivity in the development of applications, and contain special functions for the development and use of interactive applications, called dialogs. Specifically:

- ISPF is a dialog manager for interactive applications. It provides control and services to support execution of the dialogs in the MVS, VM/SP, and VSE environments.
- PDF is a facility that aids in the development of dialogs and other types of applications. It makes use of display terminals and an interactive environment to help with many programming tasks.

This manual provides a detailed description of the dialog management services and related information required to develop and use an interactive application that runs under ISPF. It is assumed that the reader is an application or systems programmer, engaged in the development of interactive programs, who is familiar with the host operating system environment; or is the user of an application running under ISPF. Users who do not develop programs may wish to read only the following parts of the manual, which describe application user functions:

Chapter 1, selected topic:  
Processing a Dialog  
Chapter 2, selected topics:  
Online Tutorial  
Split Screen  
ISPF Params Option  
Chapter 3, selected topics:  
Command Entry  
System Commands  
Terminal Keys  
Light Pen and Cursor Select  
Appendix B

The first five chapters of this manual describe how to use ISPF facilities to support the operation of interactive applications -- dialog organization, use of variables, library setup, invocation of dialogs, etc. Following that are detailed descriptions of each dialog service, and then detailed formats for panel, message, and file definitions. Finally, the appendixes contain a description of PARMs option use, an example of using the DISPLAY and TBDISPL services, a services summary, character translations for APL, TEXT, and Katakana keyboards, how to use the command table utility, and the appendix described below.



For the MVS and VM/SP operating systems, the dialog management functions of ISPF and the program development functions of PDF were previously combined in the predecessor program product, the System Productivity Facility (SPF). Major changes from SPF are described in an Appendix G.

ISPF Dialog Management Services Examples (SC34-2085) is a supplement to this manual and provides examples (in CLIST, EXEC 2, COBOL, FORTRAN, and PL/I) of ISPF application dialogs. Other related publications are:

<u>MVS</u>	<u>VM/SP</u>	<u>VSE</u>	
<u>SC34-2089</u>	<u>SC34-2090</u>	<u>SC34-2079</u>	<u>ISPF/PDF Reference</u> tells how to use the Program Development Facility.
<u>SC34-2084</u>	<u>SC34-2083</u>	<u>SC34-2080</u>	<u>ISPF and ISPF/PDF Installation and Customization</u> provides information needed to install ISPF and PDF and to custom tailor these products for a particular installation.

#### SYNTAX NOTATION

In this manual, the following notation conventions are used:

- Uppercase commands and their operands should be entered as shown. Operands shown in lowercase are variable; a value should be substituted for them.
- Operands shown in brackets [] are optional, with a choice indicated by vertical bars |. One or none may be chosen; the defaults are underscored.
- Operands shown in braces { } are alternatives; one must be chosen.
- An ellipsis (...) indicates that the parameter shown may be repeated to specify additional items of the same category.

#### TERMINOLOGY

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

- Library - A partitioned data set in the MVS environment, a MACLIB in the VM/SP environment, a VSE/AF source library in the VSE/AF 1.3.5 environment, or a VSE/AF library in the VSE/AF 2.1 environment.
- File - A sequential data set in the MVS environment, a sequential CMS file in the VM/SP environment, or a sequential data set in the VSE environment.
- Command Procedure - A CLIST in the MVS environment, or an EXEC in the VM/SP environment.

# CONTENTS

<b>Chapter 1. Introduction</b> . . . . .	<b>1</b>
Using ISPF Services . . . . .	2
Developing a Dialog . . . . .	3
An Example of a Dialog . . . . .	4
Processing a Dialog . . . . .	5
MVS and VM/SP: Invoking an ISPF Application . . . . .	5
VSE: Invoking an ISPF Application . . . . .	5
Invoking an ISPF Application from a Master Application Menu . . . . .	6
Data Communications Within ISPF . . . . .	7
Parts of a Dialog . . . . .	7
<b>Chapter 2. Dialog Processing Concepts</b> . . . . .	<b>9</b>
Dialog Organization . . . . .	9
Control Facilities . . . . .	12
MVS and VM/SP: Starting a Dialog . . . . .	12
VSE: Starting a Dialog . . . . .	12
The SELECT Service . . . . .	14
Online Tutorial . . . . .	16
Split Screen . . . . .	16
ISPF PARMS Option . . . . .	17
Dialog Services Overview . . . . .	17
Display Services . . . . .	17
Panel Definitions . . . . .	18
Message Definitions . . . . .	20
Table Services . . . . .	22
Table Residency . . . . .	23
Accessing Data . . . . .	23
General Services . . . . .	24
MVS, VM/SP, and VSE/AF 1.3.5: Resource Protection . . . . .	25
VSE/AF 2.1: Resource Protection . . . . .	25
Row Services . . . . .	26
Example . . . . .	26
Table Size . . . . .	28
File Tailoring Services . . . . .	28
Skeleton Files . . . . .	29
Example . . . . .	31
Variable Services . . . . .	32
Variable Access - Order of Search from Services . . . . .	32
Select Service and Variable Pools . . . . .	33
Relationship of Function Pools to Dialog Functions . . . . .	35
MVS and VM/SP: The Function Pool for Command Procedures . . . . .	35
The Function Pool for Programs . . . . .	36
The Shared Pool . . . . .	38
The Application Profile Pool . . . . .	38
Representation of Variables . . . . .	39
System Variables . . . . .	40
Summary of Variable Services . . . . .	42

Miscellaneous Services . . . . .	42
EDIT and BROWSE Services . . . . .	42
LOG Service . . . . .	43
CONTROL Service . . . . .	43
VM/SP: Use of the Virtual Machine Communication Facility (VMCF)	44
<b>Chapter 3. Use of Commands, Program Keys, and Light Pen . . .</b>	<b>45</b>
Command Entry . . . . .	45
System Commands . . . . .	47
STOPAT Command . . . . .	49
END and RETURN Commands . . . . .	50
Jump Function . . . . .	51
Scrolling . . . . .	52
Command Tables . . . . .	53
Command Table Format . . . . .	54
SELECT Action Commands . . . . .	55
Assigning Command Aliases . . . . .	56
Overriding System Commands . . . . .	56
Passing Commands to a Dialog Function . . . . .	57
Dynamically Specified Command Actions . . . . .	58
Terminal Keys . . . . .	59
Program Function Keys . . . . .	59
Defining PF Keys . . . . .	60
Saving PF Key Definitions . . . . .	62
MVS and VM/SP: Program Access (PA) Keys . . . . .	62
VSE: Program Access (PA) Keys . . . . .	63
Light Pen and Cursor Select . . . . .	63
<b>Chapter 4. Library Requirements . . . . .</b>	<b>65</b>
MVS: Library Setup . . . . .	65
Required Libraries . . . . .	65
Table and File Tailoring Libraries . . . . .	66
CLIST and Program Libraries . . . . .	67
VM/SP: Library Setup . . . . .	68
Required Libraries . . . . .	68
Table and File Tailoring Libraries . . . . .	69
EXEC and Program Libraries . . . . .	71
Restrictions on Use of MODULE Files . . . . .	72
MVS and VM/SP: Use of Libraries . . . . .	72
VSE/AF 1.3.5: Library Setup . . . . .	73
Required Libraries . . . . .	73
Library Definition . . . . .	74
Table and File Tailoring Libraries . . . . .	78
VSE/AF 1.3.5: Use of Libraries . . . . .	78
VSE/AF 2.1: Library Setup . . . . .	79
Required Libraries . . . . .	79
Library Definition . . . . .	80
Table and File Tailoring Libraries . . . . .	84
VSE/AF 2.1: Use of Libraries . . . . .	85
<b>Chapter 5. Invocation and Termination . . . . .</b>	<b>87</b>
MVS and VM/SP: Invocation of ISPF . . . . .	87
VSE: Invocation of ISPF . . . . .	87
ISPSTART Syntax . . . . .	88

Test Modes . . . . .	91
Trace Modes . . . . .	93
Dialog Initiation and Termination . . . . .	93
SELECT Service Invocation . . . . .	93
VSE: Dialog Abend Intercept . . . . .	95
Batch Execution of ISPF Services . . . . .	95
TSO Batch Environment . . . . .	95
Sample Batch Job . . . . .	96
Error Processing . . . . .	96
VM/SP Batch Environment . . . . .	98
Sample Batch Job . . . . .	98
Error Processing . . . . .	98
VSE Batch Environment . . . . .	100
Sample Batch Job . . . . .	100
Error Processing . . . . .	101
<b>Chapter 6. Description of Services . . . . .</b>	<b>103</b>
Invocation of Services . . . . .	103
Command Invocation . . . . .	103
VM/SP: Using the &PRESUME Statement . . . . .	105
Call Invocation . . . . .	105
Parameters . . . . .	108
Return Codes from Services . . . . .	109
MVS and VM/SP: Return Codes from Services . . . . .	110
VSE: Return Codes and Other Processing Considerations . . . . .	111
FORTRAN . . . . .	111
PL/I . . . . .	111
COBOL . . . . .	112
Services . . . . .	112
BROWSE - MVS or VM/SP: Display a Data Set or File . . . . .	113
BROWSE - VSE: Display a Library or File . . . . .	116
CONTROL - Set Processing Modes . . . . .	120
DISPLAY - Display Panels and Messages . . . . .	124
EDIT - MVS or VM/SP: EDIT a Data Set or File . . . . .	126
EDIT - VSE: EDIT a Library or File . . . . .	129
FTCLOSE - End File Tailoring . . . . .	133
FTERASE - Erase File Tailoring Output . . . . .	135
FTINCL - Include a Skeleton . . . . .	136
FTOPEN - Begin File Tailoring . . . . .	137
LOG - Write a Message to the Log File . . . . .	139
SELECT - Select a Panel or Function . . . . .	140
SETMSG - Set Next Message . . . . .	145
TBADD - Add a Row to a Table . . . . .	147
TBBOTTOM - Set the Row Pointer to Bottom . . . . .	149
TBCLOSE - Close and Save a Table . . . . .	150
TBCREATE - Create a New Table . . . . .	153
TBDELETE - Delete a Row from a Table . . . . .	155
TBDISPL - Display Table Information . . . . .	156
TBEND - Close a Table without Saving . . . . .	162
TBERASE - Erase a Table . . . . .	163
TBEXIST - Determine if a Row Exists in a Table . . . . .	164
TBGET - Retrieve a Row from a Table . . . . .	165
TBMOD - Modify a Row in a Table . . . . .	167
TBOPEN - Open a Table . . . . .	169

TBPUT - Update a Row in a Table . . . . .	171
TBQUERY - Obtain Table Information . . . . .	173
TBSARG - Define a Search Argument . . . . .	175
TBSAVE - Save a Table . . . . .	177
TBSCAN - Search A Table . . . . .	180
TBSKIP - Move the Row Pointer . . . . .	182
TBTOP - Set the Row Pointer to the Top . . . . .	184
TBVCLEAR - Clear Variables . . . . .	185
VCOPY - Create a Copy of a Variable . . . . .	186
VDEFINE - Define Function Variables . . . . .	188
VDELETE - Remove a Definition of Function Variables . . . . .	191
VGET - Retrieve Variables from a Pool or Profile . . . . .	192
VPUT - Update Variables in a Pool or Profile . . . . .	194
VREPLACE - Replace a Variable . . . . .	196
VRESET - Reset Function Variables . . . . .	198
<b>Chapter 7. Panel and Message Definition and Skeleton Formats . . . . .</b>	<b>199</b>
Panel Definitions . . . . .	199
Formatting Guidelines . . . . .	200
Syntax Rules and Restrictions . . . . .	202
General Rules . . . . .	203
Blanks and Comments . . . . .	203
Lists . . . . .	204
Variables within Text Fields and Literal Expressions . . . . .	205
Attribute Section . . . . .	206
Default Attribute Characters . . . . .	206
Statement Formats . . . . .	207
Panel Body Section . . . . .	210
Command and Message Fields . . . . .	211
Sample Body Section . . . . .	211
Model Section . . . . .	213
Initialization and Processing Sections . . . . .	213
Statement Formats . . . . .	214
Control Variables . . . . .	223
Default Cursor Positioning . . . . .	226
"Z" Variables as Field Name Placeholders . . . . .	226
Panel Processing Considerations . . . . .	227
Special Panel Requirements . . . . .	228
Menus . . . . .	229
Primary Option Menus . . . . .	232
Set Next Menu . . . . .	233
Examples of Menus . . . . .	234
Help/Tutorial Panels . . . . .	239
Table Display Panels . . . . .	243
Message Definitions . . . . .	251
Message ID . . . . .	251
Message Library . . . . .	251
Syntax Rules . . . . .	253
Skeleton Definitions . . . . .	254
Data Records . . . . .	255
Control Statements . . . . .	256
Sample Skeleton File . . . . .	258
<b>Appendix A. Using the DISPLAY Service . . . . .</b>	<b>261</b>

Reading List . . . . .	262
Steps in Function Processing . . . . .	263
Description of Steps in Function Processing . . . . .	264
<b>Appendix B. Using the ISPF PARMs Option . . . . .</b>	<b>275</b>
Specify Terminal Characteristics (Option 0.1) . . . . .	275
MVS: Specify Log and List Defaults (Option 0.2) . . . . .	279
VM/SP: Specify Console, Log, and List Defaults (Option 0.2) . . . . .	281
VSE: Specify Log and List Defaults (Option 0.2) . . . . .	282
Specify Program Function Keys (Option 0.3) . . . . .	284
<b>Appendix C. Using the TBDISPL Service . . . . .</b>	<b>289</b>
Steps in Function Processing . . . . .	290
Description of Steps in Function Processing . . . . .	291
<b>Appendix D. Command Table Utility . . . . .</b>	<b>297</b>
<b>Appendix E. Summary of ISPF Syntax . . . . .</b>	<b>301</b>
Invoking an ISPF Application . . . . .	301
Message Definitions . . . . .	301
Skeleton Control Statements . . . . .	301
Panel Definitions . . . . .	302
Panel Header Statements . . . . .	302
Attribute Section . . . . .	302
Body Section . . . . .	302
Model Section . . . . .	302
Initialization Section . . . . .	302
Processing Section . . . . .	302
Statement Specifying the End of a Panel Definition . . . . .	303
Panel Statements and Built-in Functions . . . . .	303
Attribute Section . . . . .	303
Initialization and Processing Sections . . . . .	303
Panel Control Variables . . . . .	303
Dialog Services . . . . .	304
Command Invocation Syntax . . . . .	304
Display Services . . . . .	304
Table Services - General . . . . .	304
Table Services - Row Operations . . . . .	305
File Tailoring Services . . . . .	305
Variable Services . . . . .	306
Other Services . . . . .	306
Call Invocation Syntax . . . . .	307
Display Services . . . . .	307
Table Services - General . . . . .	307
Table Services - Row Operations . . . . .	308
File Tailoring Services . . . . .	308
Variable Services . . . . .	309
Other Services . . . . .	309
<b>Appendix F. VDEFINE Exit Routine . . . . .</b>	<b>311</b>
<b>Appendix G. Character Translations for APL, TEXT, and Katakana . . . . .</b>	<b>313</b>

<b>Appendix H. MVS and VM/SP: Summary of Changes From SPF</b>	<b>317</b>
New and Revised Functions	317
Migration of Dialogs from SPF to ISPF	320
<b>Appendix I. VM/SP: Use of Shared Minidisks</b>	<b>323</b>
<b>Index</b>	<b>325</b>



# FIGURES

1.	Developing a Dialog Using PDF . . . . .	4
2.	Application Dialog Running Under ISPF . . . . .	6
3.	Typical Dialog Starting with a Menu . . . . .	10
4.	Typical Dialog Starting with a Function . . . . .	11
5.	Control and Data Flow . . . . .	13
6.	SELECT Service Used To Invoke and Process a Dialog . . . . .	15
7.	Sample Panel Definition . . . . .	19
8.	Sample Panel - When Displayed . . . . .	21
9.	Sample Member in Message Library . . . . .	22
10.	Sample Table . . . . .	23
11.	Sample Skeleton File . . . . .	30
12.	Control and Data Flow in a Dialog . . . . .	34
13.	Default Program Key Arrangement . . . . .	60
14.	PF Key Definition Panel . . . . .	61
15.	Use of Light Pen Attribute . . . . .	64
16.	ISPDEF Statement Parameters and Libraries to Which They Apply . . . . .	76
17.	Relationship Between Defaults Specified by the '*' Libname Statement and Unspecified ISPF Libraries. . . . .	77
18.	ISPDEF Statement Parameters and Libraries to Which They Apply . . . . .	82
19.	Relationship Between Defaults Specified by the '*' Libname Statement and Unspecified ISPF Libraries.Sublibraries . . . . .	83
20.	MVS Batch Job . . . . .	97
21.	VM/SP Batch Job . . . . .	99
22.	VSE/AF 1.3.5 Batch Job . . . . .	100
23.	VSE/AF 2.1 Batch Job . . . . .	101
24.	Sample Panel Definition . . . . .	212
25.	Sample Panel - When Displayed . . . . .	213
26.	Sample Panel with TRANS and TRUNC . . . . .	217
27.	Sample Panel with IF Statement . . . . .	219
28.	Sample Panel with Verification . . . . .	222
29.	Sample Panel with Control Variables . . . . .	225
30.	Example of "Z" Variable Placeholders . . . . .	227
31.	Master Application Menu . . . . .	235
32.	ISPF Primary Option Menu . . . . .	237
33.	Lower-Level Menu . . . . .	238
34.	Sample Tutorial Hierarchy . . . . .	241
35.	Sample Tutorial Panel (B) . . . . .	242
36.	Sample Tutorial Panel (F2) . . . . .	243
37.	Table Display Panel Definition . . . . .	247
38.	Current Contents of Table . . . . .	247
39.	Table as Displayed . . . . .	248
40.	Table Display Panel Definition with Multiple Model Lines . . . . .	249
41.	Table as Displayed with Multiple Model Lines . . . . .	250
42.	Sample Member in Message Library . . . . .	252
43.	Sample Skeleton File . . . . .	259
44.	Five Rows in Table Library Member TAB1 (TAB1 is Referenced by Steps 1b, 3a, 6a, and 8a) . . . . .	268

45.	Panel Library Member, Panel Definition SER (Used in Steps 2a, 4a, and 7a)	268
46.	Panel Display SER (Displayed by Steps 2a, 4a, and 7a)	269
47.	Panel Display SER With an ISPF-provided Message Superimposed on Line 1 (Displayable During Steps 2a and 7a)	270
48.	Message Library Member EMPX21 (Used by Steps 4a, 5b, and 7a)	270
49.	Panel Display SER With the Short Form of Message EMPX210 Superimposed on Line 1 (Displayed by Step 4a)	271
50.	Panel Display SER With the Long Form of Message EMPX210 Superimposed on Line 3 (Displayable During Step 4a)	272
51.	Panel Library Member, Panel Definition DATA (Used in Step 5b)	273
52.	Panel Display DATA (Displayed by Step 5b)	274
53.	Parameter Options Menu	275
54.	Terminal Characteristics Panel (MVS and VM/SP)	276
55.	Terminal Characteristics Panel (VSE)	277
56.	Log and List Defaults Panel (MVS)	279
57.	Console, Log, and List Defaults Panel (VM/SP)	281
58.	Log and List Defaults Panel (VSE)	283
59.	PF Key Definition Panel (12 PF Keys)	285
60.	PF Key Definition Panels (24 PF Keys)	284
61.	Table TAB1 Contents	294
62.	Table TAB1 as Displayed	294
63.	Table Display Panel Definition T1PANEL	295
64.	Command Table Utility Panel	298
65.	Command Table Editing Panel	299
66.	Internal Character Representations for APL Keyboards	314
67.	Internal Character Representations for Text Keyboards	315

## CHAPTER 1. INTRODUCTION

The Interactive System Productivity Facility (ISPF) Program Product provides dialog management services in three environments:

- MVS Time Sharing Option (TSO)
- VM/SP Conversational Monitor System (CMS)
- VSE/Interactive Computing and Control Facility (ICCF)

Conceptually, ISPF is an extension to these host systems. ISPF services are complementary to those of the host system and are expressly designed to implement interactive processing.

ISPF provides services to interactive applications that run under its control. Developers of these applications rely on ISPF to:

- Display predefined screen images and messages
- Originate 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 facilities (ISPF/PDF must be installed to make these facilities available), and log hardcopy output
- Control operational modes during processing

An application that runs under ISPF is called a dialog. Application coding may be done in:

- In MVS and VM/SP, a command procedure language (CLIST or EXEC 2).
- A programming language, such as PL/I, COBOL, or FORTRAN. The following compilers may be used:
  - In MVS and VM/SP:
    - PL/I Optimizer 5734-PL1
    - COBOL 5740-CB1
    - FORTRAN IV G1 5734-F02

- In VSE:
  - PL/I (Release 6) 5736-PL1
  - COBOL (Release 3) 5746-CB1
  - VS FORTRAN (Release 4) 5748-F03

A developer may use more than one language in a dialog. For example, within a single dialog containing three functions, each function could be written using a different language, such as PL/I, COBOL, and FORTRAN. In MVS and VM/SP, one (or more) of the functions could be written using a command procedure language (CLIST or EXEC 2) instead of a programming language.

A dialog coded in a programming language may be designed for cross-system use, to be processed by ISPF running under MVS, VM/SP, or VSE.

## USING ISPF SERVICES

When a dialog is running under ISPF, ISPF services are invoked from a dialog function, at the point where the service is desired, by a command (in MVS and VM/SP only) or a call statement. (In VSE, dialog functions cannot be written as ICCF procedures).

- In MVS and VM/SP, dialog functions coded in a command procedure language invoke ISPF services by means of the ISPEXEC command; for example:

*ISPEX*  
 — ISPEXEC DISPLAY PANEL(XYZ)

*see p. 103*  
 This example invokes a service that displays information on a 3270 screen (such displays are called panels). A panel definition named XYZ, prepared by the developer and prestored in the ISPF panel library, specifies both the content and the format of the display.

- Dialog functions coded in a programming language invoke ISPF services by calling a service interface routine named ISPLINK; for example, in PL/I:

*ISPLNK*  
 — CALL ISPLINK ('DISPLAY', 'XYZ');

*see p. 106*  
 This example invokes a service that displays information on a 3270 screen (such displays are called panels). ISPLINK is a small program module distributed with ISPF. (Because FORTRAN allows only six characters in a called module's name, the module may also be called by using the 6-character name ISPLNK).

In MVS and VM/SP, ISPLINK 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. Assembler programs must include code to implement the standard save area convention.

In VSE, programs issuing requests for ISPF services must adhere to the standard call, save, return linkage described in "Program Linkage" in the DOS/VSE Macro User's Guide, GC24-5139.

More examples of ISPF services requests may be found with each service description in Chapter 6, "Description of Services."

## DEVELOPING A DIALOG

A developer, using an editor such as the edit option of the ISPF/Program Development Facility (ISPF/PDF, or simply PDF), develops a dialog by entering its various components from a terminal and storing them in libraries.

Any available editor may be used by a developer when creating dialog components; however, PDF does provide special facilities to aid dialog development and testing. Figure 1 shows a developer using PDF to create and test dialog components.<sup>1</sup>

Two types of components generally present in dialogs are functions (command procedures or programs) and panel definitions. A function is created in the same way as any command procedure or program and stored in an appropriate host system library or CMS EXEC file.

A panel definition specifies a 3270 display image. The definition specifies both the content and format of a particular display. It is created by a dialog developer at a 3270 terminal and, when completed, is saved as a member of an ISPF panel library.

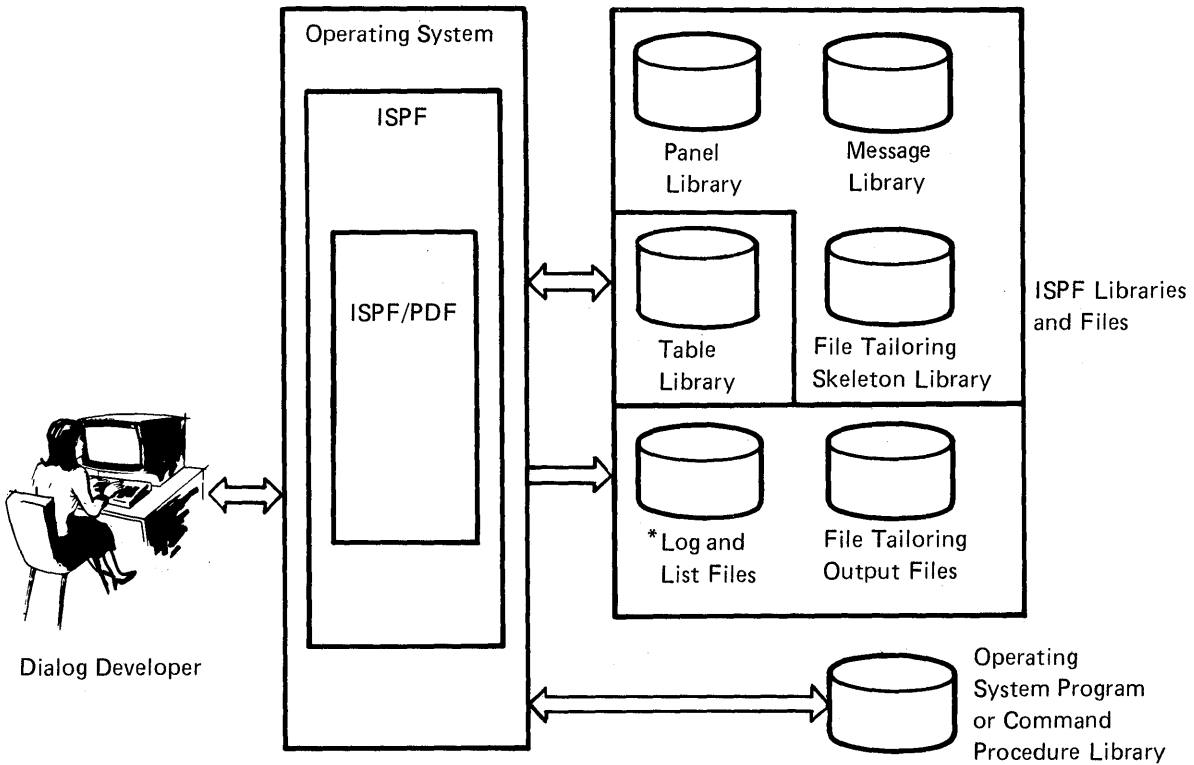
The panel definition requires no compilation or preprocessing step and closely resembles the appearance of the 3270 display screen that it specifies. Each character position in the panel definition is mapped to the same relative position on the display screen.

Panel definitions specify where variable data and literal data are to appear on the 3270 display screen. The developer specifies the place on a display where he wants a variable to appear by entering the variable's name at the same place in the panel definition. Literal data is specified by entering the literal itself on the panel definition at the place at which it is to appear on the screen.

When the developer completes the functions, panel definitions, and any other dialog components (such as messages, tables, and file tailoring

---

<sup>1</sup> Using PDF facilities to aid in dialog development and testing is described in ISPF/PDF Reference. See, particularly, "Using Dialog Development Models" and "Dialog Test (Option 7)" in that manual.



\*In addition to being an output file, the log file can be browsed and is an input file when PDF option 7.5 is in effect.

Figure 1. Developing a Dialog Using PDF

skeletons) required by the application being developed, the dialog is ready to be processed under ISPF.

#### AN EXAMPLE OF A DIALOG

An example of a dialog (consisting of a function, messages, and two display panel definitions) appears in Appendix A, "Using the DISPLAY Service" on page 261. This appendix includes a list of sections of this publication (a reading list) containing information needed to understand the example.

## PROCESSING A DIALOG

Figure 2 shows a dialog being processed under ISPF. ISPF dialog services are available only to command procedures or programs running under ISPF.

Dialog processing begins either with the display of a selection panel (also called a menu) that provides the user with a choice of actions, or with a function. The developer determines, when he designs the dialog, whether it will begin with a menu or a function.

In either case, the user can invoke a dialog from a 3270 terminal running under control of TSO, CMS, or ICCF.

## MVS and VM/SP: Invoking an ISPF Application

In MVS and VM/SP, an ISPF application is invoked from a 3270 running under TSO or CMS by use of the ISPSTART command; for example:

- ISPSTART PANEL(ABC)
  - This command invokes ISPF and specifies that dialog processing is to begin with display of a selection panel named ABC from the panel library.
- ISPSTART CMD(DEF)
  - This command invokes ISPF and specifies that dialog processing is to begin with a command procedure function (CLIST or EXEC 2) named DEF.
- ISPSTART PGM(GHI)
  - This command invokes ISPF and specifies that dialog processing is to begin with a program function named GHI.

## VSE: Invoking an ISPF Application

In VSE, an ISPF application is invoked from a 3270 running under ICCF by use of an ICCF procedure. Assuming that the name of this procedure is ISPSTART, ISPF is invoked by a single command; for example:

- ISPSTART 'PANEL(ABC)'
  - This command invokes ISPSTART, an ICCF procedure, which, in turn, invokes ISPF. 'PANEL(ABC)' specifies that dialog processing is to begin with display of a selection panel named ABC from the panel library.
- ISPSTART 'PGM(GHI) LANG(COBOL)'



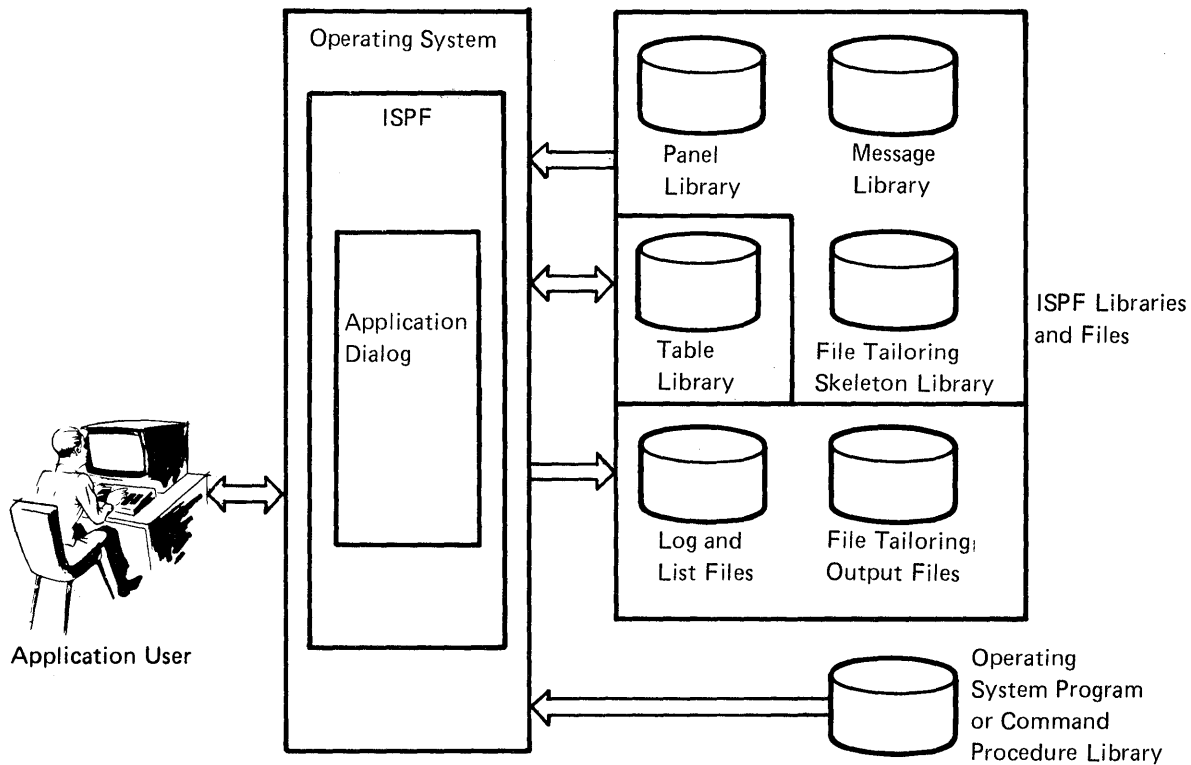


Figure 2. Application Dialog Running Under ISPF

- This command invokes ISPSTART, an ICCF procedure, which, in turn, invokes ISPF. 'PGM(GHI) LANG(COBOL)' specifies that dialog processing is to begin with a program function named GHI, which was written in the COBOL programming language.

### Invoking an ISPF Application from a Master Application Menu

At installations that provide an ISPF master application menu, the user can invoke a dialog by making appropriate selections on that menu, or on that menu and one or more subsequently displayed menus. (A master application menu is one from which any of the installation's applications may be invoked. It generally is displayed at the outset of each ISPF session.) In these installations, the master menu is generally invoked by the ISPSTART command or an ICCF procedure invoked (with no operands) at a terminal.

In MVS and VM/SP, ISPSTART may be issued automatically as part of a user's logon procedure. Also, the ISPSTART command may be issued from a command procedure (CLIST or EXEC 2).

## DATA COMMUNICATIONS WITHIN ISPF

Data is communicated within a dialog and to ISPF services by means of dialog variables. A dialog variable is a character string referred to by a symbolic name. Dialog variables may be defined and used in panels, messages, skeleton definitions, and in functions that comprise a dialog. For example, a dialog variable name can be defined in a panel definition and be referenced in a function of the same dialog. Or, the variable can be defined in a function and used in a panel definition to initialize information on a display panel, and then used to store data entered by the user on the display panel.

For functions coded in a programming language, the internal program variables that are to be used as dialog variables may be identified to ISPF through the use of the ISPF VDEFINE service, or the program may access and update dialog variables by using the ISPF VCOPY and VREPLACE services. These services do not apply to command procedures.

In MVS and VM/SP, for functions coded as a command procedure (CLIST or EXEC 2), variables used in the procedure are automatically treated as dialog variables; no special action is required to define them to ISPF.

## PARTS OF A DIALOG

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

- Functions - a function is a command procedure or a program that performs processing requested by the user. It may invoke ISPF dialog services to display panels and messages, build and maintain tables, generate output files, and control operational modes.
- Panels - a panel is a predefined display image. It may be a menu, a data entry panel, or an information-only panel. 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 and superimposed on the display to which they apply, 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 among several applications. The type and amount of data stored in a table depends upon the nature of the application.

- File Tailoring Skeletons - a file tailoring skeleton (or simply a skeleton) is a generalized representation of sequential data that 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 skeletons may not be needed, depending upon the type of application.

Panel definitions, message definitions, and 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 and updated during dialog execution. The organization of each table is specified to ISPF by the functions that use ISPF.

## CHAPTER 2. DIALOG PROCESSING CONCEPTS

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

### DIALOG ORGANIZATION

A dialog may be organized in a variety of ways to suit the requirements of the application and the needs of the application user.

A typical dialog organization, shown in Figure 3, starts with display of the highest menu in a hierarchy (called 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 menu. Each lower-level menu may also cause functions to receive control, or still other menus to be displayed. The menu hierarchy may extend as many levels deep as desired.

**Note:** Menus are also called selection panels. In this publication, the terms are used interchangeably.

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

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

Figure 4 shows another type of dialog organization in which a dialog function receives control first, prior to the display of a menu. The function 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.

As shown in Figure 4, a function may also invoke another function, using SELECT, without displaying a menu. (In MVS and VM/SP, 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 menu process, again by using the SELECT service.

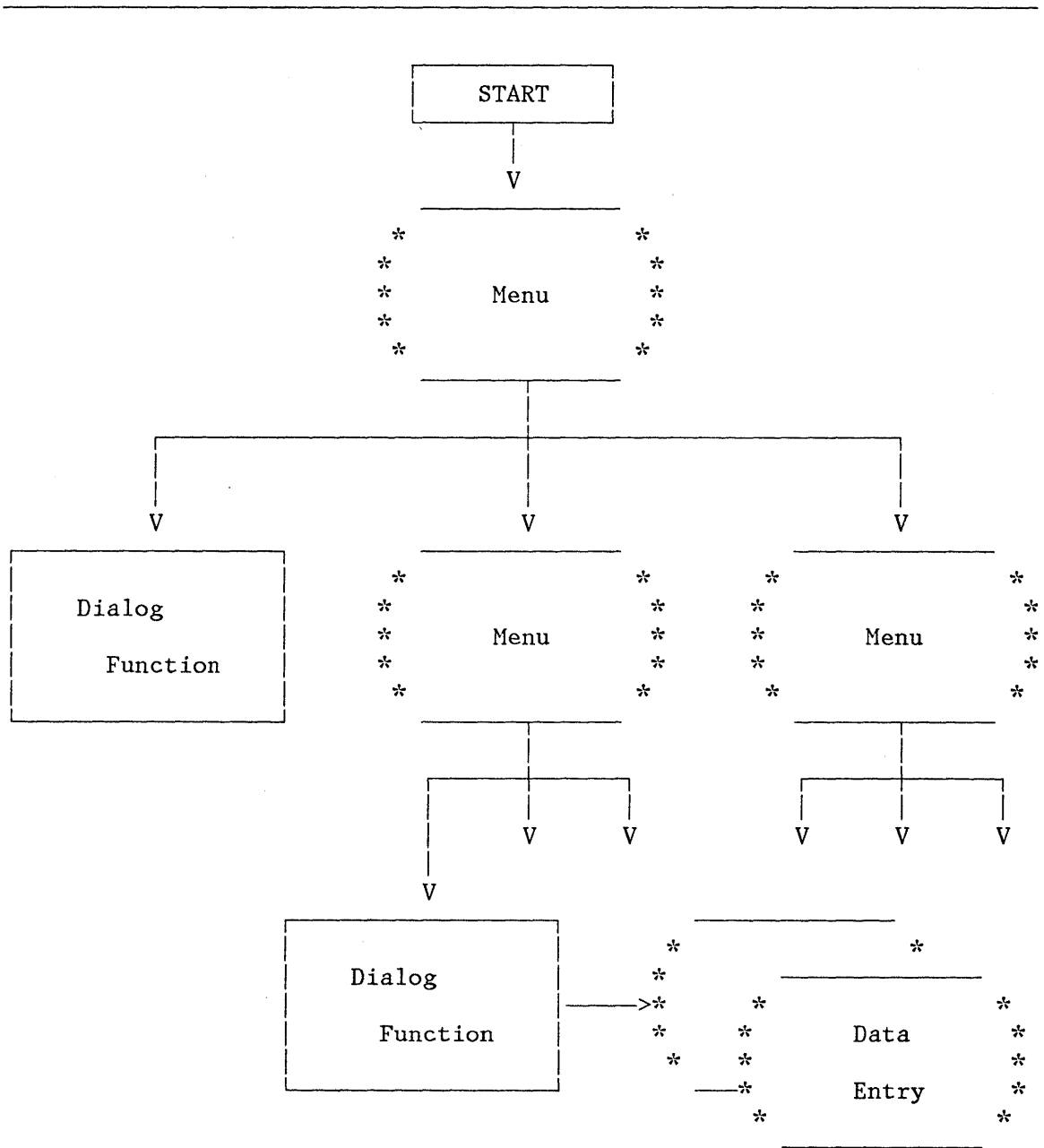


Figure 3. Typical Dialog Starting with a Menu

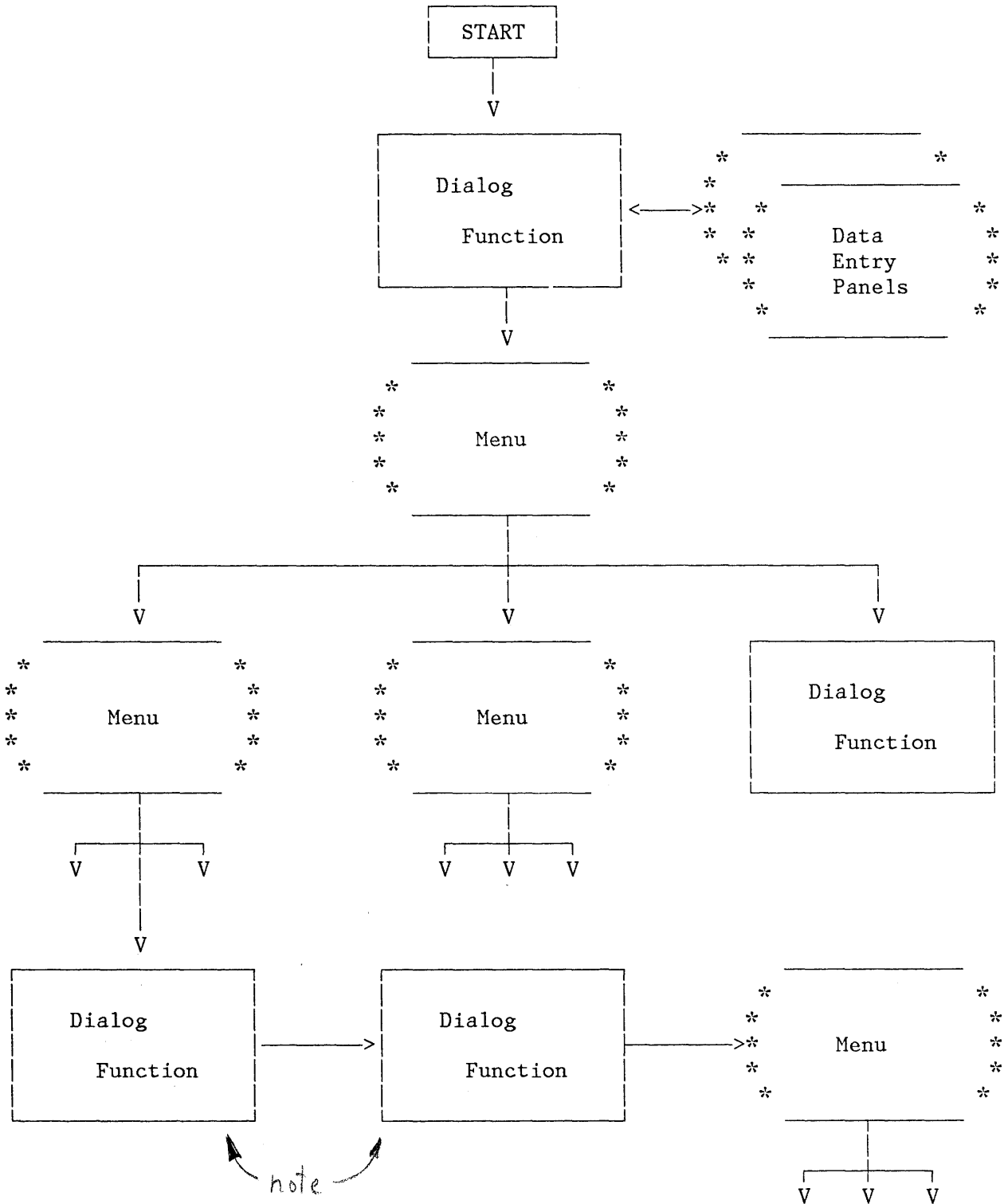


Figure 4. Typical Dialog Starting with a Function

## CONTROL FACILITIES

Control and data flow are shown in Figure 5.

A dialog is started by means of the ISPSTART command or, in VSE, by means of an ICCF procedure. ISPSTART is the recommended name for this procedure and, in this publication, that is the name used.

## MVS AND VM/SP: STARTING A DIALOG

In MVS and VM/SP, a dialog is started by the ISPSTART command. The ISPSTART command may be entered in various ways:

- By a user at the terminal
- From a command procedure (CLIST or EXEC 2)
- During LOGON (from a TSO LOGON procedure or CMS PROFILE EXEC)

ISPSTART command parameters specify the first menu to be displayed or the first dialog function to receive control (prior to the display of a menu). In this case, the ISPSTART command is typically entered during LOGON or from a command procedure.

### Example:

A user begins an application from his terminal by entering the ABC command. ABC allocates the appropriate libraries for the application, and then issues an ISPSTART command to begin ISPF processing. Thus, the ABC command serves as a "front end" to start the application. ABC cannot use ISPF dialog services, because it does not run under ISPF.

## VSE: STARTING A DIALOG

In VSE, a dialog is started by an ICCF procedure, generally named ISPSTART. This procedure may be invoked directly from a terminal. A parameter, passed in the command that invokes the ISPSTART procedure, specifies the first menu to be displayed or the first dialog function to receive control (prior to display of any menu).

### Example:

A user invokes an application from his terminal by invoking an ICCF procedure named ISPSTART. This procedure defines libraries for the application and invokes ISPF. The procedure cannot use ISPF dialog services, because it does not run under ISPF.



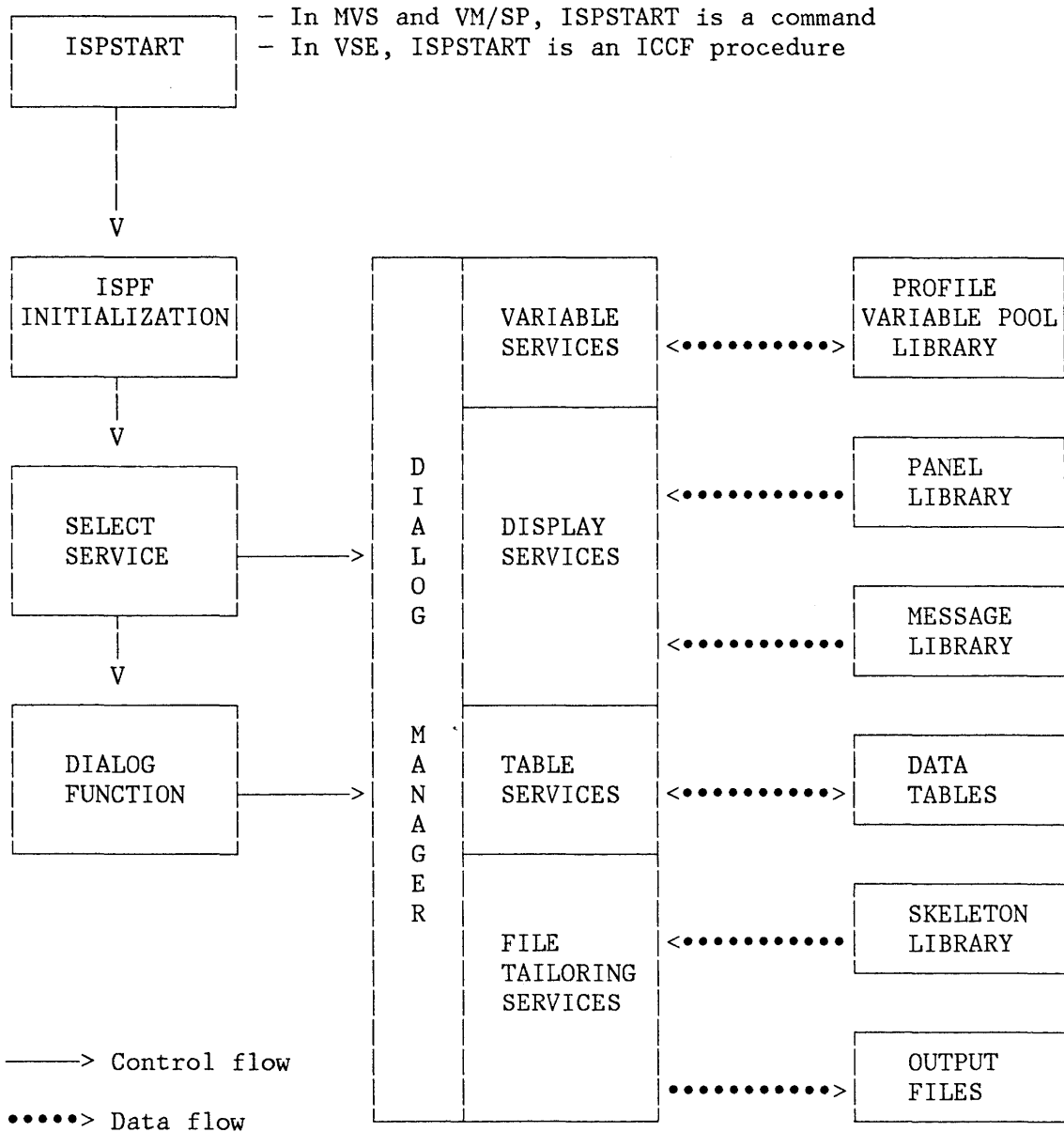


Figure 5. Control and Data Flow

## THE SELECT SERVICE

SELECT is both a control facility and a dialog service. SELECT is used by ISPF during its initialization to invoke a function or selection panel that begins a dialog.

During dialog processing, SELECT may be used in the dialog to display menus and invoke program or command procedure functions.

Parameters passed to SELECT specify the next action as follows (CMD applies only in MVS and VM/SP; in VSE, dialog functions may not be written as ICCF procedures):

PANEL(panel-name)  
CMD(command)  
PGM(program-name)

The PANEL parameter specifies the name of the next panel to be displayed. The CMD and PGM parameters specify a dialog function (coded as a command or program, respectively) to receive control. Input parameters may be passed to the dialog function as part of the command specification or, for programs, by the use of the PARM parameter.

Figure 6 emphasizes that the SELECT service is used when invoking and when processing a dialog. After SELECT is used to start a dialog, it is used by the dialog, as a dialog service, to invoke a function or to display a menu. In turn, that function or menu may use SELECT to invoke another function or to display another menu. This function or menu may, in turn, using SELECT, invoke still another function or menu. This process may continue for many levels and establishes a hierarchy of invoked functions and menus. There is no restriction as to the number of levels allowed in this hierarchy.

When a lower-level function or menu in the hierarchy completes its processing, control is returned to the higher-level function or menu from which it was invoked. The higher-level function resumes its processing or the higher-level menu is redisplayed for the user to make another selection. Thus, SELECT is used in a dialog to establish a hierarchy of functions and menus and this hierarchy determines the sequence in which dialog functions and menus are processed, including the sequence in which they are terminated.

In MVS and VM/SP, dialog functions written as command procedures may directly invoke other functions written as command procedures without using the SELECT service.

Program-coded functions can invoke another function only through using the SELECT service. Thus when a program-coded function calls another program directly, without using the SELECT service, the called program is treated as part of the function that called it. It is not treated as a new dialog function by ISPF.

The scope of a dialog function is defined as the period of time from the invocation of a function to its termination.

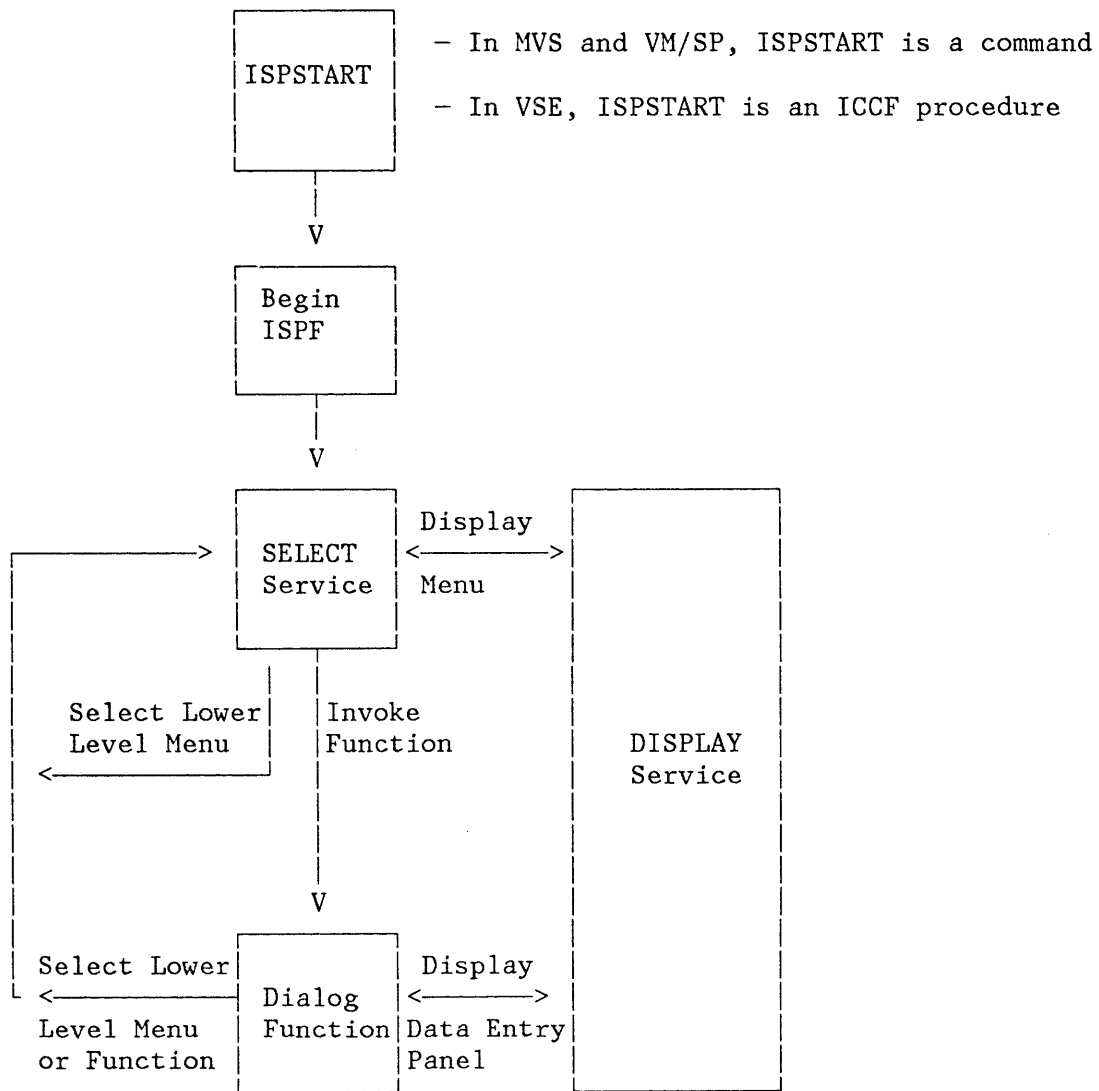


Figure 6. SELECT Service Used To Invoke and Process a Dialog

## ONLINE TUTORIAL

A tutorial is a set of panels that provides online information for a user of an application. A dialog developer may include tutorial panels with his dialog to be displayed at the option of the user. Generally, the developer incorporates information that is helpful to a first-time user or is instructive about the actions the user may select when some particular condition occurs during application processing.

The program that displays tutorial panels is part of ISPF. It may be entered in either of two ways:

- As an option from a menu
- Indirectly from any non-tutorial panel when the user enters the HELP command

Transfer in and out of the tutorial using the HELP command 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 by invoking the HELP command, the first panel to be displayed is a panel within the hierarchy, appropriate to what the user was doing when the HELP command was invoked.

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

BACK or B - to return to the previously viewed tutorial 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 ends the tutorial, by means of the END or RETURN command, the panel from which the tutorial was entered is redisplayed.

## SPLIT SCREEN

At any time during a dialog, the user may partition the display screen into two "logical" screens. The two logical screens are treated as though they were independent terminals. ISPF provides control for mapping the two logical screens onto the physical screen. (Panels that are displayed by the DISPLAY service always pertain to a logical screen.)

In split screen mode, only one of the logical screens is considered active at a time. The location of the cursor identifies the active screen.

Use of split screen mode and the size of each logical screen is under control of the user and is transparent to the dialog function.

In VSE, user dialog functions are restricted to one logical screen. While a user dialog function may be executed on either logical screen, it may not be executed on both logical screens concurrently. This restriction does not apply to ISPF and ISPF/PDF functions.

Split screen mode is entered by invoking the SPLIT command. This command is also used to reposition the split line that separates the two logical screens. Thus, the size of a logical screen may be changed by use of the SPLIT command. Split screen mode is terminated by ending the application on either logical screen. The remaining logical screen is then expanded to its full size.

## ISPF PARMS OPTION

ISPF includes a facility, generally referred to as "ISPF PARMS," that allows a user to specify terminal characteristics, default options for processing the ISPF list and log files, and program function (PF) key assignments. This facility operates as a dialog and may be invoked from other dialogs. Typically, it should be included as an option on an application's primary option menu. See "Primary Option Menu" on page 235 for an example of how to specify the invocation of the ISPF PARMS option. See Appendix B, "Using the ISPF PARMS Option" for a description of how to use the PARMS option.

## DIALOG SERVICES OVERVIEW

The display services, table services, file tailoring services, variable services, and miscellaneous services available to dialogs are described in the following sections.

### Display Services

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

- DISPLAY - Display a panel
- TBDISPL - Display a table
- SETMSG - Display a message on the next panel

The DISPLAY service reads panel definitions from the panel library, initializes variable information in the panel from the corresponding dialog variables in the function, shared, or profile pools, and displays

the panel on the screen. Optionally, a message may be superimposed on the panel display.

After the user has entered information, it is stored in the corresponding dialog variables in the function, shared, or profile pools, and the DISPLAY service returns to the calling function.

Use of the DISPLAY service is illustrated in Appendix A, "Using the DISPLAY Service."

The TBDISPL service combines information from panel definitions with information stored in ISPF tables. It displays selected columns from a table, and allows the user to identify rows for processing. The user may scroll the table information up and down (see "Scrolling" in Chapter 3, "Use of Commands, Program Keys, and Light Pen").

Panel definitions used by the table display service contain non-scrollable text, including column headings, followed by one or more "model lines" that specify how each row from the table is to be formatted in the scrollable area of the display.

Use of the TBDISPL service is illustrated in Appendix C, "Using the TBDISPL Service."

The SETMSG service constructs a specified message in a system save area. The message will be superimposed on the next panel displayed by any ISPF service.

## Panel Definitions

A panel definition consists of up to five sections:

- 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, which may be overridden, are provided.
- Body section (required) - defines the format of the panel as seen by the user, and defines the name of each variable field on the panel.
- Model section (required for table display; not allowed for other types of panels) - specifies the format for displaying each row of the table.
- Initialization section (optional) - specifies the initial processing that is to occur prior to displaying the panel. This section is typically used to define how variables are to be initialized.
- Processing section (optional) - specifies processing that is to occur after the panel has been displayed. This section typically is used to define how variables are to be verified and/or translated.

The panel definition syntax is fully described in Chapter 7, "Panel and Message Definition and Skeleton Formats."

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 7. 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
```

---

```
)BODY
%----- EMPLOYEE RECORDS -----+
%COMMAND ==>_ZCMD +
%
%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 7. Sample Panel Definition

---

For text fields, the information following the attribute character ("% or "+") is the text to be displayed. Text fields may contain substitutable variables consisting of an ampersand (&) followed by a dialog variable name. The ampersand and name 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 Figure 7 has eleven input fields (ZCMD, TYPECHG, LNAME, etc.), indicated with underscores. It also has a substitutable variable (EMPSER) within a text field (on line 5). The first three lines of the panel and the arrows preceding the input fields are all highlighted, as indicated by percent signs. The other text fields are low intensity, as indicated by plus signs.

Before the panel is displayed, all variables in the panel body are automatically initialized from the corresponding dialog variables (ZCMD, TYPECHG, LNAME, etc.) and EMPSER. After the panel has been displayed and the user has finished entering requested input and depresses the ENTER key, the input fields are automatically stored into the corresponding dialog variables.

Figure 8 shows the panel as it appears when displayed, assuming that the current value of EMPSER is "123456", and that the other variables are initially null or blank. Panel variables that are blank are initialized to null.

## Message Definitions

Message definitions are created by dialog developers, using an editor, and are saved in a member of 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 in the message library consists of two lines. The first line contains the message id, and optionally:

- The short message text, enclosed in apostrophes (')
- The name of the corresponding help panel (if the user requests help when the message is displayed)
- The audible alarm indicator (yes or no)

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

Short messages are displayed in the upper right-hand corner of the screen. If the user enters the HELP command (or presses the Help PF key), the long message is then displayed on line 3 of the screen. If the user requests help again, tutorial mode is entered.

**Note:** The default screen positions for short and long messages (upper right-hand corner and line 3, respectively) may be changed by a dialog developer specifying some other position in the panel definition.

If a short message is not specified, the long message is displayed first. If the user then requests help, tutorial mode is entered.

Variable names, preceded by an ampersand (&), may appear anywhere within the short and long message text, and are replaced in the display with their current value.

---

```
----- EMPLOYEE RECORDS -----
COMMAND ==>

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 8. Sample Panel - When Displayed

---

Figure 9 shows an example of a member in the message library. This member contains all messages that begin with a message id of "EMPX21". The message definition syntax is fully described in "Message Definitions" in Chapter 7, "Panel and Message Definition and Skeleton Formats."

*Page 251* }

---

```

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 '&EMP SER ADDED'
'EMPLOYEE &LNAME, &FNAME &I ADDED TO FILE.'

EMPX218 '&EMP SER UPDATED'
'RECORDS FOR &LNAME, &FNAME &I UPDATED.'

EMPX219 '&EMP SER DELETED'
'RECORDS FOR &LNAME, &FNAME &I DELETED.'

```

Figure 9. Sample Member in Message Library

---

## Table Services

Table services allow sets of dialog variables to be maintained and accessed. A table is a 2-dimensional array of information in which each column corresponds to a dialog variable, and each row contains a set of values for those variables.

A table is shown in Figure 10. In this example, the variables that define the columns are:

```

EMP SER - 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	Robertson	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 10. Sample Table

---

### Table Residency

A table may be either temporary or permanent. A temporary table exists only in virtual storage, it cannot be written to disk storage. A permanent table, while created in virtual storage, may be saved 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. When a table is being updated (in virtual storage), the copy of the table on direct access storage cannot be accessed until the update has been completed.

Permanent tables are maintained in one or more table libraries, in which each member contains an entire table.

For both temporary and permanent tables, rows are accessed and updated from the in-storage copy. A permanent table that has been accessed read-only may be modified in virtual storage, but may not be written back to disk storage.

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 Chapter 4, "Library Requirements," for a discussion of libraries.

### Accessing Data

The variable names that define the columns of a table are specified when the table is created. Each variable is specified as either a KEY field or a NAME (non-key) field. One or more columns (variable names) may be specified as keys for accessing the table. For the table shown in Figure 10, 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 and 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 use of the "current row pointer" (CRP). The table may be scanned by moving the CRP forward or back. A row is retrieved each time the CRP is moved. When a table is opened, the CRP is automatically positioned to TOP -- ahead of the first row.

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 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. These extension variables, and their values, are added to the row. This permits variables to be stored in the row that were not specified when the table was created. A list of extension variable names for a row may be obtained when the row is read. If the list of extension variables is not respecified when the row is rewritten, the extensions are deleted.

## General Services

The following services operate on an entire table:

**TBCREATE** Creates a new table and opens it for processing

**TBOPEN** Opens an existing (permanent) table for processing

**TBQUERY** Obtains information about a table

**TBSAVE** Saves a permanent copy of a table without closing

**TBCLOSE** Closes a table, and saves a permanent copy if the table was opened in WRITE mode

**TBEND** Closes a table without saving

**TBERASE** Deletes a permanent table from the table output 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. The table 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.

## MVS, VM/SP, and VSE/AF 1.3.5: Resource Protection

Table services provides a resource protection mechanism designed to prevent concurrent updating of the same table by multiple users. This protection mechanism is built on the assumption that all users who may want to update a given table will have the same first library definition for ISPTLIB.

When a table is opened or created in write mode, an exclusive enqueue (MVS and CMS) or lock (VSE) is requested for a resource name consisting of the first library name in the ISPTLIB definition concatenated with the table name. The TBOpen or TBCreate service fails with a return code of 12 if this enqueue or lock is unsuccessful. A successful enqueue or lock stays in effect until the completion of a TBend or TBClose service for the table. If the NAME parameter is specified on the TBsave or TBClose service, an additional exclusive enqueue or lock is issued. The resource name consists of the first library name in the ISPTLIB definition concatenated with the name specified in the NAME parameter. If this enqueue or lock fails, the service terminates with a return code of 12 and the table is not written.

The table output library represented by the ISPTABL definition is protected from concurrent output operations from any ISPF function through a separate mechanism not specific to table services. Volume protection prevents physically writing to the same volume by more than one user at a time.

## VSE/AF 2.1: Resource Protection

Table services provides a resource protection mechanism designed to prevent concurrent updating of the same table by multiple users. This protection mechanism is built on the assumption that all users who may want to update a given table will have the same first library definition for ISPTLIB.

When a table is opened or created in write mode, an exclusive lock is requested for a resource name consisting of the first library.sublibrary name in the ISPTLIB definition concatenated with the table name. The TBOpen or TBCreate service fails with a return code of 12 if this lock is unsuccessful. A successful lock stays in effect until the completion of a TBend or TBClose service for the table. If the NAME parameter is specified on the TBsave or TBClose service, an additional exclusive lock is issued. The resource name consists of the first library.sublibrary name in the ISPTLIB definition concatenated with the name specified in the NAME parameter. If this lock fails, the service terminates with a return code of 12 and the table is not written.

The table output library represented by the ISPTABL definition is protected from concurrent output operations from any ISPF function through a separate mechanism not specific to table services. Volume protection prevents physically writing to the same volume by more than one user at a time.

## Row Services

The following services operate on a row of the table:

- TBADD** Adds a new row to the table
- TBDELETE** Deletes a row from the table
- TBGET** Retrieves a row from the table
- TBPUT** Updates an existing row in the table
- TBMOD** Updates a row in the table if it exists (if the keys match); otherwise, adds a new row to the table
- TBEXIST** Tests for the existence of a row (by key)
- TBSCAN** Searches a table for a row that matches a list of "argument" variables, and retrieves the row
- TBSARG** Establishes a search argument for use with TBSCAN
- TBTOP** Sets CRP to TOP (ahead of the first row)
- TBBOTTOM** Sets CRP to the last row and retrieves the row
- TBSKIP** Moves the CRP forward or back by a specified number of rows, and then retrieves the row at which the CRP is positioned
- TBVCLEAR** Sets dialog variables (that correspond to variables in the table) to null

## Example

The following series of commands demonstrate the use of table services (also see Appendix C, "Using the TBDISPL Service"):

1. Create a permanent table, named DALPHA, to consist of dialog variables AA, BB, and CC. AA is the key field; BB and CC are name fields.

```
ISPEXEC TBCREATE DALPHA KEYS(AA) NAMES(BB CC) WRITE
```

```
DALPHA
```

AA	BB	CC

2. Display a panel named XYZ. (This panel requests a user to enter values for dialog variables AA, BB, and CC, which are used in following steps of this example.)

**ISPEXEC DISPLAY PANEL (XYZ)**

3. Assume the user enters the following values on panel XYZ:

AA = Pauly John  
 BB = W590  
 CC = Jones Beach

ISPF automatically updates dialog variables AA, BB, and CC -- in the function variable pool -- with the user-entered values.

Record these values in the table DALPHA.

**ISPEXEC TBADD DALPHA**

DALPHA

AA	BB	CC
Pauly John	W590	Jones Beach

4. Assume the following values for dialog variables AA, BB, and CC are entered by a user, as in step 2, through a panel display operation:

AA = Clark Joan  
 BB = Y200  
 CC = Bar Harbor

Record these values in table DALPHA.

**ISPEXEC TBADD DALPHA**

DALPHA

AA	BB	CC
Pauly John	W590	Jones Beach
Clark Joan	Y200	Bar Harbor

Table services adds a row to table DALPHA immediately following the row added by the previous TBADD. Following the TBADD, the current row pointer (CRP) is positioned at the newly added row. (Before a row is added by the the TBADD service, table service scans the table to determine if the KEY field of the new row to be added duplicates



the KEY field of an existing row. If it does, the TBADD is not performed.)

5. Save table DALPHA (by writing it to the table output library) for later use.

### ISPEXEC TBCLOSE DALPHA

The table DALPHA is written from virtual storage to member DALPHA in the table library.

### Table Size

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

$$\text{Row size} = 22 + 4a + b + 9c$$

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:

$$\text{DTCB length} = 152 + 16d$$

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 ISPF. The name of the temporary file is available in system variable ZTEMPF. In MVS, ZTEMPF contains a data set name; in VM/SP and VSE, ZTEMPF contains a filename.

## Skeleton Files


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 from a variable pool (see "Variable Access - Order of Search from Services").

Skeleton file records may also contain statements that control processing. These statements 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

When iteratively processing records, file tailoring services read each row from a specified table. If the table was already open (prior to processing the skeleton), it remains open with the CRP positioned at TOP. If the table was not already open, file tailoring opens it automatically and closes it upon completion of processing.

A sample skeleton file is shown in Figure 11. 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/SP environment, it could be punched to an MVS machine for batch execution.



---

```

//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 11. Sample Skeleton File

---

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** Prepares the file tailoring process, and specifies whether the temporary file is to be used for output.

**FTINCL** Specifies the skeleton to be used, and starts the tailoring process.

**FTCLOSE** Ends the file tailoring process.

**FTERASE** Erases (deletes) an output file created by file tailoring.

### Example

The following example illustrates file tailoring services. For this example assume that:

- **LABLSKEL** is a member in the file tailoring skeleton library, containing:

**LABLSKEL**

```
)DOT DALPHA
  NAME: &AA
  APARTMENT: &BB
  CITY: &CC
  YEAR: &ZYEAR
)ENDDOT
```

**ZYEAR** is the name of an ISPF system variable that contains the current year.

- **DALPHA** is a member in the table library, containing:

**DALPHA**

AA	BB	CC
Pauly John	W590	Jones Beach
Clark Joan	Y200	Bar Harbor

1. Issue ISPF commands to process skeleton **LABLSKEL**. Obtain values for dialog variables **AA**, **BB**, and **CC** from table **DALPHA**. The resulting file tailoring output consists of one address label for each row of information in table **DALPHA**.

```
ISPEXEC FTOPEN
ISPEXEC FTINCL LABLSKEL
```

**FTOPEN** prepares for access (opens) both the file tailoring skeleton and file tailoring output libraries. These libraries must have been allocated before starting the ISPF session.

**FTINCL** performs the file tailoring process using the file tailoring skeleton named **LABLSKEL**. **LABLSKEL** contains the file tailoring controls, **)DOT** and **)ENDDOT**, which specify the use of table **DALPHA**.

2. Write the resulting file tailoring output to a member named LABLOUT in the file tailoring output library

ISPEXEC FTCLCLOSE NAME (LABLOUT)

3. At the conclusion of processing the above commands, file tailoring output library member LABLOUT contains:

LABLOUT

NAME: Pauly John
APARTMENT: W590
CITY: Jones Beach
YEAR: 82
NAME: Clark Joan
APARTMENT: Y200
CITY: Bar Harbor
YEAR: 82

## Variable Services

Variable services allow the definition and use of dialog variables. Dialog variables are the main communication vehicle between dialog functions (program modules or command procedures) and ISPF services. Program modules, command procedures, panels, messages, tables, and skeletons can all reference the same data through the use of dialog variables.

A dialog variable's value is a character string that may vary in length from zero to 32K bytes. Some services restrict the length of dialog variable data. For example, dialog variables used as input or output fields in panels are limited to a length of 255 bytes.

Dialog variables are referenced symbolically, by name. The name is composed of one to eight characters (six, for FORTRAN). Alphameric characters (A-Z, 0-9, #, \$, or @) are used in the name, but the first character may not be numeric.

### Variable Access - Order of Search from Services

Dialog variables are organized into groups (or pools) according to the dialog function and application with which they are associated. (See the SELECT service for a description of how the dialog developer can control creation of these pools.)

A pool may be thought of as a list of variable names that enables ISPF to access the associated values. When an ISPF service encounters a dialog variable name (in a panel, message, table, or skeleton) it searches these pools to access the dialog variable's value. The pools and the types of dialog variables that reside in them are shown below in the standard search sequence used by ISPF services.

search first

search last

1. Function pool - A variable that resides in the function pool of the function currently in control is called a function variable. It is accessible only by that function. ("lost" after next CLIST/EXEC)
2. Shared pool - A variable that resides in the shared pool of the current application is called a shared variable. It is accessible only by functions belonging to the same application.
3. Application profile pool - A variable that resides in the application profile pool is called an application profile variable and is automatically retained for the user from one session to another. Profile variables are automatically available when an application begins and are automatically saved when it ends.

### Select Service and Variable Pools

Figure 12 shows how the SELECT service may be used to pass control within a dialog and illustrates the resulting variable pool structures. The flow is shown from the initial command entry through menus and through dialog functions. ISPF access to dialog variables is shown at each point.

Initially, menus A and B are processed. Since menus are not part of any function, all variables are accessed from the shared and profile pools. Function X is invoked and uses the VPUT service to copy one of the variables from its function pool into the shared pool. Next, function Y is invoked. Function Y is shown using the VGET service to copy the dialog variable from the shared pool to its function pool and then using the SELECT service for further menu processing.

(not FUNCTION pool)

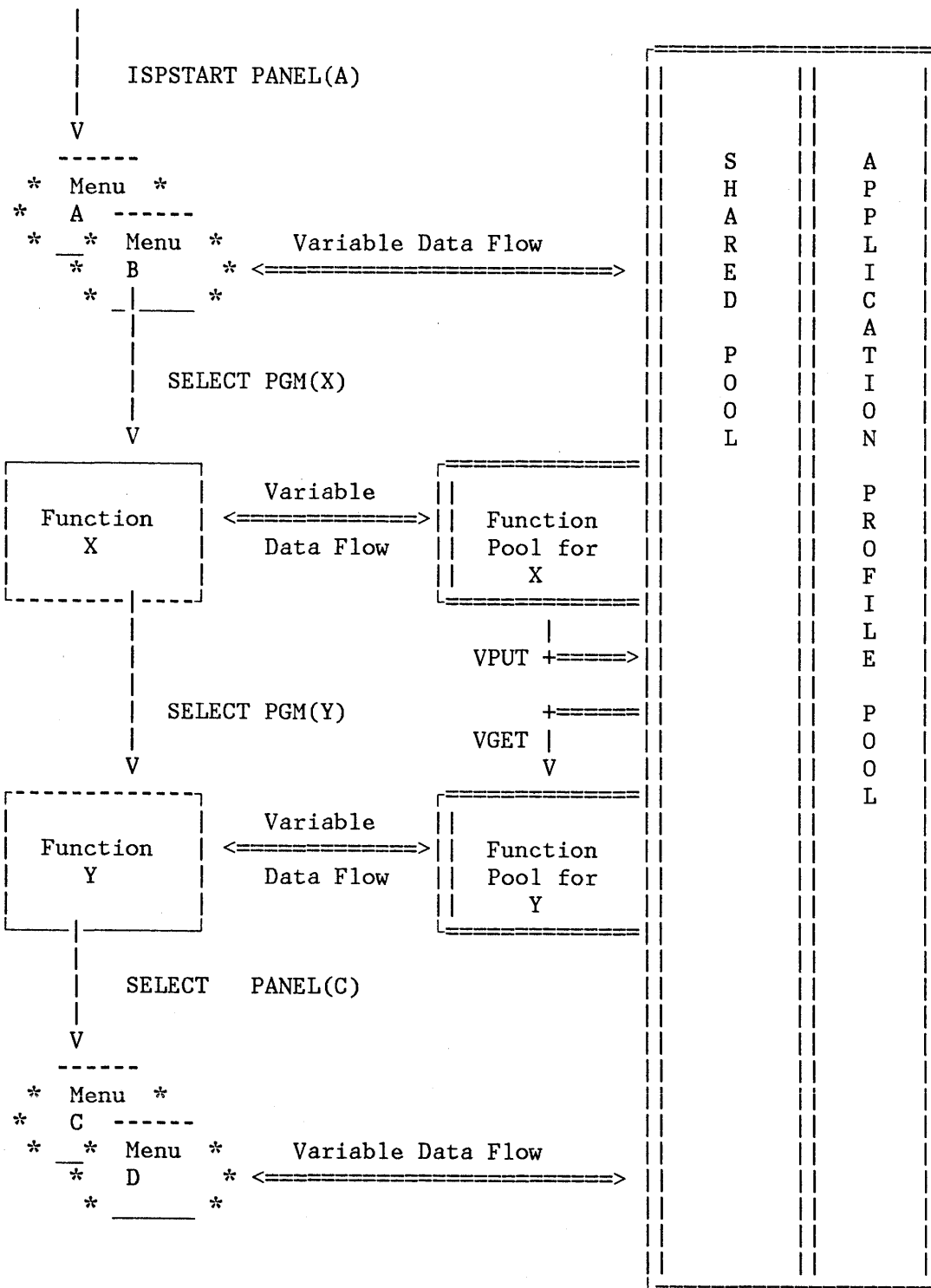


Figure 12. Control and Data Flow in a Dialog

Figure 12 also shows how the SELECT service controls access to dialog variable pools from both functions and menus.

When a variable is defined as an input variable on a menu, the following actions apply during processing of the menu:

- If the variable does not exist in either the shared pool or the profile, it is created in the shared pool.
- If the variable exists in the shared pool, it is accessed from, and is updated in, the shared pool.
- If the variable exists in the profile and not in the shared pool, it is accessed from, and is updated in, the profile pool.

### Relationship of Function Pools to Dialog Functions

Each dialog function has associated with it a unique function pool of dialog variables. The pool is maintained by ISPF for the associated function. The function uses the dialog variables to communicate with the various ISPF services.

When a dialog variable is created, it is entered into the current function pool and may not be referenced by other functions. (Dialog variables associated with one function may have the same names as dialog variables associated with another function, but they are not the same variables.)

When a new function is started, a function pool is created for it. Variables may then be created in the function pool and accessed from it. When the function ends, its function pool -- along with any variables in it -- is deleted.

### MVS and VM/SP: The Function Pool for Command Procedures

In MVS and VM/SP, when the function in control is a command procedure (CLIST or EXEC 2), the list of variable names kept by the command language processor and the list of function variables kept by ISPF are the same list. Thus, a variable created by the command procedure during its execution is automatically a dialog variable, and a dialog variable entered in the function pool by ISPF is automatically accessible to the command procedure. However, in ISPF, variable names may not exceed eight characters.

#### **Notes:**

1. EXEC 2 variables &DATE and &TIME and CLIST variables &SYSDATE and &SYSTIME may not be used in ISPF. Use instead, ISPF system variables ZDATE and ZTIME, which contain similar information.
2. In MVS, TSO global variables in effect at when ISPF is started, are not available to CLISTs running under ISPF. These global variables



are restored when ISPF terminates. Any global variables put into effect from within ISPF are lost when ISPF terminates.

The following command procedure example (CLIST language) illustrates that command procedure variables are automatically treated as ISPF dialog variables.

**Example:**

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

In the example above, variable AAA is created by the command procedure simply by setting it to a value (a "1", in this case). The DISPLAY service is then invoked to display panel XYZ.

Assume that the panel definition for XYZ contains two dialog variables, named AAA and BBB, and that they are defined as input (unprotected) fields. In the panel definition, they might appear as follows:

```
+ INITIAL VALUE %====>_AAA      +
+ INCREMENT     %====>_BBB      +
```

where the underscore indicates the start of an input field, followed by the name of the variable.

When the panel is displayed, the content of AAA (a "1") is displayed. ISPF creates the variable BBB in the function pool and displays it as a blank.

Now assume the user, in response to the panel display, types 100 in the first field (AAA) and types 20 in the second field (BBB). When the user presses the ENTER key, the value 100 is automatically stored in AAA and the value of 20 is automatically stored into BBB. The DISPLAY service then returns control to the command procedure. Then the next statement is executed. This statement creates variable CCC, setting it to the sum of AAA and BBB, namely 120.

The Function Pool for Programs

When the dialog function in control is a program, the program does not share a common list of variables with ISPF because a program is compiled, not interpreted as command procedures are. ISPF maintains a list of variables that belong to the function so that ISPF services can use dialog variables for communication of data. ISPF makes two types of entries in the function pool for a program: defined and implicit.

DEFINED VARIABLES: Defined variables are entered explicitly through the use of the VDEFINE service. The VDEFINE service creates a dialog variable name in the function pool and associates it with the program's own variable. This association enables ISPF to directly access and modify that program variable. Otherwise, the program's variables are

not available to ISPF. The VDELETE service ends this association and removes ISPF's ability to access that program variable.

Example: The following program code (PL/I) specifies that field PA of the program can be accessed by ISPF using dialog variable name FA.

```
DECLARE PA CHAR(8);
DECLARE LENGTHPA FIXED BIN(31) INIT(LENGTH(PA));
PA = 'OLD DATA';
CALL ISPLINK ('VDEFINE', 'FA ', PA, 'CHAR', LENGTHPA);
CALL ISPLINK ('DISPLAY', 'XYZ ');
```

PA is declared as a program variable (character string, length 8). The program calls the VDEFINE service to make PA accessible to ISPF by use of dialog variable FA. Then, the DISPLAY service is called to display panel XYZ. Assuming dialog variable FA is specified as an input field on the panel definition, then when data is entered for field FA, ISPF stores the data into the program variable PA.

**IMPLICIT VARIABLES:** Implicit variables are placed in the function pool when either of the following circumstances arise:

1. When an ISPF service refers to a dialog variable name that is not found in the standard search reference.
2. When an ISPF service must store into a dialog variable that does not already exist in the function pool.

Implicit variables can not be accessed directly from a program function. Programs access implicit variables only through the use of VCOPY and VREPLACE. However, any ISPF service invoked by a program function may access an implicit variable directly by referencing the variable name.

To illustrate how an implicit variable is created, assume that panel XYZ, in the above example, allows the user to enter another value and the panel definition specifies that this value is to be stored in dialog variable IA. This is the first reference to IA and therefore it does not exist in the function pool. Because IA does not exist, ISPF creates it in the function pool and stores into IA the value entered on the panel by the user. IA is an implicit dialog variable.

**IDENTICAL DIALOG VARIABLE NAMES:** A defined variable and an implicit variable can have the same name. This occurs when (using the VDEFINE service) a defined variable is created using the same name as an existing implicit variable. When the same name exists in both the defined and the implicit areas of a function pool, the defined entry can be accessed and the implicit entry can not be accessed. The implicit entry is made accessible by removing (through VDELETE) any entries for that variable name made through the VDEFINE service.

A given dialog variable name can be defined (by VDEFINE) many times within a given function. Each definition may associate a different program variable with the dialog variable name. This is referred to as "stacking." When each successive VDEFINE request is processed for a

given dialog variable name, the previous definition is not accessible. Only the most recent definition of that dialog variable is accessible. A previous definition of that variable may be made accessible by deleting (using VDELETE) the more recent definitions of that name.

For example, the mainline of a program can define a dialog variable to be associated with one program variable. A subroutine is called and can define the same dialog variable name to be associated with a different program variable. Any ISPF services invoked after the second VDEFINE would have access to only the subroutine's program variable. The subroutine would delete (using VDELETE) that dialog variable before returning, thereby uncovering the earlier definition set up in the mainline program.

**Note:** To avoid a possible program error, for each VDEFINE processed within a function for a given dialog variable name, a VDELETE should be processed using the same name.

### The Shared Pool

The shared variable pool allows functions and selection panels to share access to dialog variables.

Shared pools are created by the SELECT service when it processes the ISPSTART or ISPF command and when the NEWAPPL or NEWPOOL keywords are specified with the SELECT service. When SELECT returns, the shared pool is deleted and the previous shared pool (if any) is reinstated.

A function may copy dialog variables from its function pool to the shared pool by means of the VPUT service. Since a panel displayed by the SELECT service does not belong to any function, any dialog variables used in the panel are read from and stored into the shared or profile pools.

Variables in the shared pool are accessible to all ISPF services that use the standard search sequence. In addition, another function may directly copy these variables to its function pool by means of the VGET service.

### The Application Profile Pool

Like the shared variable pool, the application profile pool contains variables that are accessible to functions within an application, but the profile variables are saved across sessions. (An application consists of one or more dialogs, each of which have been started using the same application id).

When a new application is started, it has access to an application profile variable pool. If an application is restarted (for example, by split screen) then both invocations of the application access exactly the same application profile. The profile data is maintained as an ISPF table whose name is xxxxPROF, where xxxx is the application-id. If the

application is already active, then the current profile is used. Otherwise, ISPF must search for the table.

When accessing an application profile that is not currently active, ISPF first searches the user's profile library for a member with the name xxxPROF. The member is found if the user had previously run the application and so had a local copy of the profile table.

If the member is not found, the table input library is searched. The application developer may provide a profile in this library. This profile is to contain variable names and values initialized for the application.

If the member cannot be found in either the user's profile or table input library, the application profile is initialized with the contents of the default application profile pool, ISPPROF, which is read from the table input library. ISPPROF is distributed with ISPF and contains a set of default PF key values. An installation may modify this table to change these settings or to include other variables which will be copied to initialize brand new application profiles. Refer to Chapter 4, "Library Requirements," for information on profile and table libraries.

Upon completion of the application, the contents of the application profile pool are saved in the user's profile library under the name xxxPROF. The profile is deleted from storage when the last invocation of the application is terminated.

Functions can directly access the profile pool using the VGET variable services. The VPUT service must be used to enter variables in the profile pool. However, menus (selection panels) automatically update existing profile variables.

**Note:** A second level of profile variable pool, the system profile pool (ISPSPROF), is always active. These dialog variables are owned by the dialog manager and may not be modified by an application. Their values may be read, however, because the system profile is included in the standard search sequence after the application profile. All system variable names begin with "Z" (such as "ZTERM") and supply information such as terminal type and list/log defaults.

### 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, in a profile, or in ISPF tables.

Defined variables, however, may be translated to fixed binary or to a bit string, hex string, or a user-defined format when stored internally in a program module. The internal format is specified when the variable is defined (through the use of the VDEFINE service). The translation

occurs automatically when the variable is stored by an ISPF 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, the remaining data will be lost.
- Translation - if the variable is defined as other than a character string, and the external representation has invalid characters, the contents of the defined variable is lost.

In either case, the ISPF 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". Therefore, dialog developers should avoid names which begin with "Z" when choosing dialog variable names. System variables are used to communicate special information between the dialog and the dialog manager. The variables are discussed with the ISPF service to which they apply.

Some system variables cannot be modified. They provide the dialog with information about the environment, such as user id, current date, and time. These variables reside in the shared variable pool. They may be obtained for a command function through the VGET service, and for a program function through the VCOPY service.

Commonly used system variables that a dialog may access are listed below:

**Note:** \* = may not be modified by a dialog

#### General

- \* ZUSER - User id
  - \* ZPREFIX - TSO user prefix in MVS; in VM/SP and VSE, ZPREFIX contains the same value as ZUSER
  - \* ZLOGON - Stepname of TSO LOGON procedure in MVS; in VM/SP and VSE, ZLOGON contains a null value
  - \* ZTIME - Time of day (format hh:mm)
  - \* ZDATE - Current date (format yy/mm/dd)
  - \* ZJDATE - Day-of-year date (format yy.ddd)
  - \* ZDAY - Day of month (2 characters)
  - \* ZMONTH - Month of year (2 characters)
  - \* ZYEAR - Year (2 characters)
  - \* ZTEMPF - Name of temporary file for file tailoring output
  - \* ZAPPLID - Application identifier
  - \* Z - Null Variable
- 
- ZTERM - Terminal type
  - ZKEYS - Number of PF keys
  - ZPFxx - Setting for PF Keys:
    - ZPF13-ZPF24 contain settings for the primary keys
    - ZPF01-ZPF12 contain settings (on 24-key terminals only) for the alternate keys
  - ZERRMSG - Error message id
  - ZERRSM - Short error message text
  - ZERRLM - Long error message text
  - ZERRHM - Name of help panel associated with error message
  - ZVERB - Command verb after a command table (SETVERB) action
  - ZDTOP - Current top row upon return from table display
  - ZSCBR - Scroll amount for the BROWSE service
  - ZSCED - Scroll amount for the EDIT service
  - ZSCML - Scroll amount for member lists

#### Tutorial Panels

- ZUP - Name of parent panel
- ZCONT - Name of next continuation panel
- ZIND - YES specifies an index page
- ZHTOP - Name of top panel
- ZHINDEX - Name of first index panel

#### Selection Panels

- ZCMD - Command input field
- ZSEL - Command input field truncated at first period
- ZPARENT - Parent menu name (when in explicit chain mode)
- ZPRIM - YES specifies panel is a primary option menu

## Summary of Variable Services

The variable services are:

- value in EXECs  
(and elsewhere)* {
- VGET Retrieve variables from shared pool or profile
  - VPUT Update variables in shared pool or profile
- NOT from EXECs* {
- VDEFINE Define function variables
  - VDELETE Remove definition of function variables
  - VRESET Reset function variables
  - VCOPY Copy data from a dialog variable to the program
  - VREPLACE Copy data from the program to a dialog variable

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

## Miscellaneous Services

ISPF provides EDIT, BROWSE, LOG, and CONTROL services. These services are discussed below. EDIT and BROWSE are available only if ISPF/PDF is installed.

### EDIT and BROWSE Services

The EDIT and BROWSE services allow a dialog function to invoke the ISPF/PDF edit or browse programs. These services require specification of a data set name (MVS), fileid (VM/SP), or filename (VSE) and a member name, if applicable. The entry panel, which is displayed if edit or browse is selected from the primary option menu, is bypassed. See ISPF/PDF Reference.

In MVS, EDIT and BROWSE services use subpools 2 and 3 and will issue FREEPOOL macros for the subpools that they use. Therefore, a dialog that invokes EDIT and BROWSE services should not use subpools 2 and 3. EDIT and BROWSE services must not be invoked from PL/I programs that also use subtasking.

## LOG Service

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

## CONTROL Service

The CONTROL service allows a dialog function to condition ISPF to expect certain kinds of display output, or to control the disposition of errors encountered by ISPF services. For example, some display conditions are:

- see p. 120*
- LINE** Expect line output, to be generated by the dialog or by execution of a TSO or CMS command. In MVS, optionally, the starting line may be specified. In VM/SP and VSE, the starting line is ignored.
- LOCK** Allow the next display without unlocking the 3270 keyboard. LOCK is generally used with the DISPLAY service to overlay a currently displayed panel with an "in process" message; for example:
- ```
CONTROL DISPLAY LOCK
DISPLAY MSG (message-id)
```
- SPLIT** Enable or disable split screen operation by a user as required by the application.
- REFRESH** Refresh the entire screen on the next display. Typically used before or after invoking some other full screen application that is not using ISPF display services.
- NONDISPL** Do not display the next panel (process the panel without actually displaying it, and simulate the ENTER key or END command).

The disposition of errors may be controlled as follows:

- CANCEL** Terminate the dialog function on an 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 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 through the SELECT service, nor a higher-level function when the current function completes.



## VM/SP: USE OF THE VIRTUAL MACHINE COMMUNICATION FACILITY (VMCF)

For VM/SP systems, the Virtual Machine Communication Facility (VMCF) is used by ISPF and may be used by dialogs. (VMCF provides services to allow virtual machines to communicate messages and data. ISPF uses VMCF in its implementation of ENQ/DEQ services to control file sharing by dialogs.)

To use VMCF facilities,<sup>2</sup> a dialog must issue the following, in the sequence shown, for each transaction:

1. VMCF diagnose - to authorize the virtual machine for general communications (see note)
2. CMS HNDEXT SET macro and STCTL/LCTL instructions - to enable the virtual machine for external interrupts

**Note:** Because ISPF services may change the settings for VMCF authorization and interrupt handling, these settings must be reestablished following each use of ISPF services.

3. VMCF diagnose - for data transfer protocol
4. CMS HNDEXT CLR macro and STCTL/LCTL instructions - to disable the virtual machine for external interrupts; issued at the end of transaction processing and before returning.

In addition, the dialog:

- Must accept the IDENTIFY protocol as well as any desired data transfer protocols.
- Must not use the VMCF AUTHORIZE SPECIFIC or VMCF UNAUTHORIZE services. Their use could result in an ABEND due to an ISPF ENQ/DEQ failure.

---

<sup>2</sup> Refer to VM/SP CMS Command and Macro Reference Manual, SC19-6209, and VM/SP System Programmer's Guide, SC19-6203 for more information.

## CHAPTER 3. USE OF COMMANDS, PROGRAM KEYS, AND LIGHT PEN

This chapter describes the three levels of ISPF commands and their use and processing, and the operation of the program access (PA) and program function (PF) keys of the 3270 terminal.

Commands may be used to request processing functions. There are three levels of commands:

- System commands - provided by the dialog manager and always available to an end user (unless explicitly overridden by an application).
- Application commands - available to an end user throughout operation of an application.
- Function commands - meaningful only while operating a particular function within an application.

The first two levels (system and application commands) are defined through the use of command tables. Processing of these commands is handled by the dialog manager. Use of system and application commands is generally transparent to the dialog functions. For example, HELP is a system command.

The third level (function commands) includes all commands that are processed by a dialog function. For example, the edit NUMBER command is a function command.

**Note:** Virtual machine settings (such as device spool and tag settings and minidisk concatenation) could be changed during display of a panel by execution of CP or CMS commands or SELECT. For example, the terminal user may change the spooling of the printer to class C by entering on the command line of a displayed panel:

```
CP SPOOL PRT CLASS C
```

Therefore, upon return from ISPF services, the dialog should establish any settings required for its processing.

### COMMAND ENTRY

The user may enter a command by:

- Pressing a PF key.

- Selecting an ATTENTION FIELD by using the light pen or the cursor select key. (The cursor select key is a hardware feature on 3278 and 3279 terminals.)
- Typing the information in the command field and pressing ENTER. (This includes the command field in browse, edit, member lists, and table displays, as well as the command field on a panel.)

ISPF intercepts all commands entered by the user, regardless of whether the command was typed in the command field or entered with a PF key or attention field. If the command matches an entry in the application or system command table, it is executed by the dialog manager. Otherwise, it is assumed to be a function command and is passed through to the dialog function. See "Command Tables" for more information on how ISPF intercepts and processes commands.

Commands may be passed to the operating system by entering the appropriate ISPF-provided command verb (CP, TSO, CMS, or ICCF) followed by the actual TSO, VM/SP, or ICCF command; for example:

```
====> TSO LISTC LEVEL(Z77PHJ)
====> CMS L * * A
====> ICCF /LIST XYZ
```

Commands may be stacked for execution by entering a special delimiter between the commands; for example:

```
====> SORT BLDG DEPT NAME; MENU ABC
```

**Note:** Commands cannot be stacked following the HELP command (HELP processing deletes any commands in the stack).

The default delimiter is a semicolon (;), which the user may change with the ISPF parms option (see Appendix B, "Using the ISPF PARMs Option").

In the above example, the SORT command is executed first. When it completes, the MENU command is executed.

**Note:** In VSE, do not attempt to execute ICCF commands from both screens at once, because ICCF can process just one command at a time.

**PRESSING A PF KEY:** In ISPF, the PF keys have not been assigned to special functions. Each PF key is equated to a character string and simply simulates command entry. When the PF key is pressed, the processing is the same as though the character string had been typed in the command field and the ENTER key pressed. A dialog function cannot sense the difference between a command entered by a PF key and a command entered by typing in the command field.

When a PF key is pressed, there may be user-entered information already in the command field. If so, the PF key definition, followed by a blank, is concatenated ahead of the contents of the command field. For example, suppose PF7 is equated to the character string "UP". If the

user types "4" in the command field and then presses PF7, the results are exactly the same as if the user had typed "UP 4" in the command field and then pressed ENTER.

A command entered in the command field is passed to the function in control if it is logically not related to the function of the PF key depressed. However, any valid commands chained behind an unrelated command are concatenated with the PF key definition.

**SELECTING AN ATTENTION FIELD:** Attention fields may also be used to simulate command entry. When an attention field is detected by the light pen or cursor select key, the processing is exactly the same as though the contents of the attention field had been typed into the command field and the ENTER key pressed. Again, the dialog function cannot sense the difference.

**Note:** Attention fields are intended as an alternative means of selecting options from a menu. They should not be used on data entry panels, since any information that is typed by the user into an input field (including command fields) will be lost when the attention occurs. Unlike PF keys, information in the command field is not concatenated with the contents of the attention field.

## SYSTEM COMMANDS

The system commands, distributed with ISPF, include the following (PF key defaults are shown in parentheses):

- |                         |                                                                                                                                                       |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>HELP (PF1/13)</b>    | Displays additional information about an error message or provides tutorial information about commands and options.                                   |
| <b>SPLIT (PF2/14)</b>   | Causes split screen mode to be entered, or changes the location of the split line.                                                                    |
| <b>END (PF3/15)</b>     | Terminates the current operation and returns to the previous menu. If the primary option menu is displayed, this command terminates ISPF.             |
| <b>RETURN (PF4/16)</b>  | Causes an immediate return to the primary option menu or to the display from which the user entered a nested dialog. (See "END and RETURN Commands.") |
| <b>RFIND (PF5/17)</b>   | Repeats the action of the previous FIND command or the FIND part of the most recent CHANGE command (applies to browse and edit only).                 |
| <b>RCHANGE (PF6/18)</b> | Repeats the action of the previous CHANGE command (applies to edit only).                                                                             |
| <b>UP (PF7/19)</b>      | Causes a scroll toward the top of the data.                                                                                                           |

|                         |                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DOWN (PF8/20)</b>    | Causes a scroll toward the bottom of the data.                                                                                                                                                                                                                                                                                                                                                  |
| <b>SWAP (PF9/21)</b>    | Moves the cursor to wherever it was previously positioned on the other logical screen of a split screen pair.<br><br>When operating in split screen mode and the SWAP key (PF9) is depressed, any entry on the command line is ignored and is not processed.                                                                                                                                    |
| <b>LEFT (PF10/22)</b>   | Causes a scroll left.                                                                                                                                                                                                                                                                                                                                                                           |
| <b>RIGHT (PF11/23)</b>  | Causes a scroll right.                                                                                                                                                                                                                                                                                                                                                                          |
| <b>CURSOR (PF12/24)</b> | Moves the cursor to the first input field on the panel being displayed (which generally is the option selection or command field) or to the alternate command field if one has been designated on the )BODY statement. If invoked a second time on a panel with scrollable data (a BROWSE, EDIT, or table display panel), this command causes the cursor to be moved to the second input field. |

The system commands described below have no default PF key assignments:

|                       |                                                                                                                                                                                                                                                                   |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>CP or CMS</b>      | In VM/SP, allows the user to enter a VM/SP command or an EXEC. Because ISPF uses the same mechanism to execute both CP and CMS commands, the two commands (CP and CMS) are treated synonymously; ISPF does not attempt to distinguish between them.               |
| <b>TSO</b>            | In MVS, allows the user to enter a TSO command or CLIST.                                                                                                                                                                                                          |
| <b>ICCF</b>           | In VSE, allows the user to enter an ICCF command.                                                                                                                                                                                                                 |
| <b>STOPAT</b>         | In VM/SP, causes display of a message and entry to CMS SUBSET mode whenever a specified program is loaded by the select service.                                                                                                                                  |
| <b>KEYS</b>           | causes an immediate display of a panel that allows the user to view and change the current PF key definitions (equivalent to option 0.3 in the ISPF PARMs option).                                                                                                |
| <b>PANELID ON OFF</b> | The command PANELID or PANELID ON causes all subsequent panels to be displayed with the name of the panel shown at the beginning of line 1. The command PANELID OFF turns off this mode of operation. During initial entry to ISPF, this mode is initialized OFF. |

INDEX  
PFSHOW

**Note:** The panel id is displayed only if the panel contains a protected-field attribute byte in row 1 column 1, and no other attribute bytes in the next eight character positions.

**PRINT** Causes a "snapshot" of the screen image to be recorded in the ISPF list file for subsequent printing.

**PRINT-HI** Same as PRINT except that high-intensity characters on the screen are printed with overstrikes to simulate the dual intensity display.

If desired, an installation may add new system-wide command definitions by modifying the system command table.

### STOPAT Command

For VM/SP systems, the STOPAT command is used to cause entry to CMS SUBSET mode when a specified program is loaded by the SELECT service. The STOPAT command is entered on the command line of a panel displayed on a logical screen:

```
COMMAND ==> STOPAT pgmname
```

where pgmname is the name of a program. Whenever the specified program is loaded by the SELECT service for the logical screen on which STOPAT was entered, a message is displayed and CMS SUBSET mode is entered. The message gives the name of the program and the storage address at which the program was loaded. For example, suppose the following STOPAT command, specifying a program named MYPROGA, is entered on the command line of a panel:

```
COMMAND ==> STOPAT MYPROGA
```

After MYPROGA is loaded by the SELECT service (assuming the storage address at which it is loaded is 000D17A0) the message displayed is:

```
PROGRAM MYPROGA  
LOADED AT 000D17A0  
CMS SUBSET
```

While in CMS SUBSET mode, the user may enter CP or CMS commands for debugging. To end CMS SUBSET mode and begin execution of the program (MYPROGA in this example), the user enters RETURN.

STOPAT is effective for one program per logical screen and remains in effect for the specified program until another STOPAT command specifying another program name is entered or a STOPAT command without a program name is entered; for example:

```
COMMAND ==> STOPAT
```

This STOPAT command, without a program name specified, cancels the STOPAT command previously in effect.

## END and RETURN Commands

The END command is used to request termination of a function or dialog. When entered on a selection panel displayed by the SELECT service, it causes a redisplay of the next higher menu in the hierarchy. When entered on a panel displayed by the tutorial program, it terminates the tutorial and causes a redisplay of the menu from which the tutorial was invoked or the panel from which HELP was requested.

When the END command is entered on a panel that was displayed by a dialog function (through the DISPLAY or TBDISPL service), the dialog function must take whatever action is appropriate to terminate and return control. Entry of the END command is signalled by a return code of 8 from the DISPLAY or TBDISPL service.

The RETURN command simulates repeated END commands, up to some appropriate stopping point, without displaying intervening panels. When a RETURN command is entered, the dialog manager takes the following action:

1. The END command is simulated on the panel that is currently displayed (i.e., the DISPLAY or TBDISPL service returns a code of 8).
2. For subsequent requests (made through the DISPLAY or TBDISPL service) for display of a different panel, the panel is not displayed and a return code of 8 is issued by the service.
3. However, when two consecutive display requests name the same panel, normal operation of the DISPLAY and TBDISPL services is restored and processing may proceed as though RETURN had not been entered. Whether to proceed is decided by the dialog developer. (Generally, because RETURN signals the application user's desire to end the current processing, a developer will limit processing, after the RETURN is received, to clean up and final processing before returning control to the dialog element from which the function was invoked.)
4. If two consecutive requests do not specify the same panel, processing continues in the mode described in item 2 above, until control is returned to a primary option menu or a nested dialog completes. Then, normal operation of the DISPLAY and TBDISPL services is restored.

**Note:** If it is necessary to suspend processing of a panel temporarily so that other panels may be displayed, issue a CONTROL DISPLAY SAVE request to save the contents and control information of the panel whose processing is to be suspended. Before resuming the processing of this panel, issue CONTROL DISPLAY RESTORE to reinstate the contents and control information for the panel. If

non-ISPF screens have been displayed, issue CONTROL DISPLAY REFRESH to clear the screen.

This mode of operation continues until a primary option menu is encountered or a nested dialog completes, whichever occurs first. If a primary option menu is encountered, it is displayed. If a nested dialog completes, the panel from which it was invoked is redisplayed exactly as the user last saw it (except that the command field is blank). In either case, this completes the action of the RETURN command.

**Note:** A nested dialog is one that is invoked from any panel by a SELECT action command (see "Command Tables"). The HELP and KEYS commands invoke nested dialogs. In addition, the TSO, CMS, and CP system commands invoke nested dialogs when they are used to execute a CLIST or EXEC that displays panels through ISPF services.

If a dialog function needs to distinguish between END and RETURN, it can do so in one of the following ways:

- Upon return from the DISPLAY or TDBDISPL service (with a return code of 8), the function may examine variable ZVERB in the shared variable pool. It will contain either "END" or "RETURN".
- Upon return from the SELECT service when the PANEL keyword was specified, the dialog function may examine the return code from SELECT. Return code 0 indicates that the END command was entered on the selected menu panel. Return code 4 indicates that the RETURN command was entered on the selected menu panel or on some lower-level menu.

## Jump Function

The jump function (also referred to as extended return) allows a user to go directly to any option that is valid from the primary option menu currently in effect.

The function is entered in the command field of any panel, preceded by an equal sign; for example:

```
COMMAND ==> =3.1
```

The action is as follows:

- If not entered on a primary option menu, the jump function causes repeated END commands to be simulated until a primary option menu is encountered. What follows the equal sign is then entered on the primary option menu and the ENTER key is simulated. (The primary option menu is not displayed.)
- If entered on a primary option menu, a jump function is treated the same as if the equal sign were not present; i.e., the specified option is selected.



Unlike the RETURN command, the jump function is unaffected by nested dialogs. For example: from the ISPF/PDF edit option, the user enters a HELP command to enter the tutorial. Then from the tutorial, the user enters "=1". This causes tutorial to end, edit to end, and primary option 1 to be invoked.

For user convenience, the jump function may be entered in any field that is preceded by an arrow. The arrow must consist of at least two equal signs followed by a greater-than sign ("==>") and must immediately precede the input attribute byte.

For compatibility with the SPF Program Product, the jump function may be entered in conjunction with the RETURN command (or RETURN PF key). For example: the user types "=2" and then presses the RETURN PF key rather than pressing ENTER. The action is just the same as if the user had typed "=2" and pressed ENTER.

When operating in split screen mode, if a user enters a jump function (for example, =3) and chains other commands to it (=3;other), the chained commands are ignored.

## Scrolling

The scroll commands 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 commands are interpreted as follows:

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

When scrollable data is displayed, scrolling allows the screen "window" to be moved up, down, left, or right across the information. (Only up and down scrolling is allowed for table displays.)

Whenever scrolling is allowed, a scroll amount is displayed at the top of the screen (line 2). This amount determines the number of lines (or columns) scrolled with each use of a scroll command. To change the scroll amount, move the cursor to the scroll field and overwrite the displayed amount. Valid scroll amounts are:

- A number from 1 to 9999 - specifies the number of lines (up or down) or columns (left or right) to be scrolled.
- PAGE - specifies scrolling by one page.
- HALF - specifies scrolling by a half page.
- MAX - specifies scrolling to the top, bottom, left margin, or right margin, depending upon which scrolling command is used.

- CUR - specifies scrolling based on the current position of the cursor. The line or column indicated by the cursor is moved to the top, bottom, left margin, or right margin of the screen, depending upon which scrolling command is used. If the cursor is not in the body of the data, or if it is already positioned at the top, bottom, left margin, or right margin, a full page scroll occurs.

For scrolling purposes, a "page" is defined as the amount of information currently visible on the logical screen. In split screen mode, for example, a browse display might have 12 lines by 80 columns of scrollable data. In this case, a scroll amount of HALF would move the text up or down by 6 lines, or right or left by 40 columns.

The current scroll amount is saved in the application profile. Three values are saved -- one for browse (ZSCBR), one for edit (ZSCED), and one for member lists (ZSCML). When you overtype the scroll amount, the new value remains in effect until you change it again. The value MAX is an exception; following a MAX scroll, the scroll amount reverts to its previous value.

Users can also enter any valid scroll amount as part of the scroll command. For example, enter:

COMMAND ==> up 3

and press the ENTER key, or enter

COMMAND ==> 3

and press the UP PF key. Either form results in a temporary, one-time override of the scroll amount.

## COMMAND TABLES

System and application commands are implemented through the use of command tables.

A system command table (ISPCMDS) is distributed with ISPF in the table input library. An application may provide an application command table by including a table named xxxxCMDS in its table input library, where xxxx is a 1- to 4-character application id.

ISPTLIB

Whenever a command is entered, the dialog manager searches the application command table (if any) and then the system command table. If the command is found, action is taken immediately. If the command is not found in the application or system tables, the command is passed through to the dialog, unaltered, in the command field. The dialog may then take appropriate action.

## Command Table Format

(ISPCMD5 TABLE I)  
to: A-disk

A command table is an ISPF table in which each row contains the specification for one command. (Refer to Appendix D, "Command Table Utility," for a discussion of the utility that is used to generate or modify command tables.) The variables that define the table columns are:

- ZCTVERB - specifies the name of the command. A command name must be from two to eight characters in length, and must begin with an alphabetic character. Note that the terms 'command name' and 'command verb' are synonymous and are used interchangeably.
- ZCTTRUNC - specifies the minimum number of characters that the user must enter to find a match with the command name. If this number is zero or equal to the length of the name, the user must enter the entire name. This number may not be one, nor greater than the length of the name.
- ZCTACT - specifies the action to be performed when the command is entered. See below.
- ZCTDESC - contains a brief description of the purpose of the command. This variable is optional. It is not used by the dialog manager in processing the command, but it is displayed by the command table utility. The description is limited to 57 characters.

The variable names listed above (ZCTVERB, ZCTTRUNC, ZCTACT, and ZCTDESC) are treated as defined function variables by the dialog manager; they are not accessible to dialogs. ZCTACT defines the action that the command specified in ZCTVERB will perform.

The valid actions that can be specified in the ZCTACT variable are:

- SELECT (followed by selection keywords) - causes the selected dialog (command, program, or menu) to be given control immediately.
- ALIAS (followed by the name of another command) - allows specification of command aliases.
- PASSTHRU - causes the command to be passed to the dialog, as though it had not been found in the table.
- SETVERB - causes the command to be passed to the dialog with the command verb stored separately from the parameters.
- NOP - causes the command to be functionless. An "inactive command" message is displayed in this case.
- Blank (no action) - causes the table entry to be ignored, and scanning to continue (to search for additional entries with the same name).

- A variable name (beginning with an ampersand) whose content may be one of the above valid actions - allows dynamic specification of command action.

These actions are described in the sections that follow.

*not to be NOPed  
PASSTHRU !?*

Additional action keywords are used to indicate system commands for which special processing is required. These are: SPLIT, SWAP, CURSOR, PRINT, and PRINT-HI. Although these are valid actions, they are intended for use only in the system command table distributed with ISPF. They are not intended for use in application command tables.

### SELECT Action Commands

A SELECT action command is one type of command that may be specified in a command table. The action is coded exactly the same as for the SELECT service. All SELECT keywords are valid, including NEWAPPL.

The selected dialog (function or menu panel) is invoked immediately when a SELECT action command is entered on the command line of any panel. This temporarily suspends the current dialog. When the selected dialog completes, the screen is refreshed and the suspended dialog resumes execution.

Examples of SELECT action commands:

| ZCTVERB | ZCTTRUNC | ZCTACT                           |
|---------|----------|----------------------------------|
| SORT    | 0        | SELECT PGM(PQRSORT) PARM(&ZPARM) |
| PREPARE | 4        | SELECT CMD(XPREP &ZPARM) NEWPOOL |
| MENU    | 4        | SELECT PANEL(&ZPARM)             |

**Note:** In VSE, CMD is not used.

In the example, the TRUNC variable indicates that the SORT and MENU command names may not be truncated. PREPARE, however, may be truncated to any of the following: PREPAR, PREPA, PREP. The functions and keywords in the ZCTACT field indicate the actions that the commands will perform.

The ZPARM variable that appears in the SELECT keywords indicates that command parameters are to be substituted at that point. For example, if the following commands were entered:

```
====> SORT BLDG DEPT NAME
====> PREPA LOG LISTING
====> MENU PQRMENU1
```

the following SELECT actions would result:

```
SELECT PGM(PQRSORT) PARM(BLDG DEPT NAME)
SELECT CMD(XPREP LOG LISTING) NEWPOOL
SELECT PANEL(PQRMENU1)
```

The ZPARM variable is used only to substitute user-entered parameters into SELECT action commands. It is a dummy variable; it is not stored in a variable pool and is not accessible to dialogs.

**Note:** Use of SELECT action commands may cause recursive entry into dialog functions, which the dialog manager allows. The dialog developer should either design functions for recursive use, or display a message if a user attempts to reenter a non-recursive function.

The ISPF EDIT and BROWSE services are non-recursive. There is an ISPF restriction that commands entered from edit and browse may not re-invoke edit and browse.

The ISPF DISPLAY and TBDISPL services may be used recursively. The current display environment is automatically saved whenever a SELECT action command is entered, and restored upon completion of the command.

### Assigning Command Aliases

A command table may establish aliases by designating, as an action in the ZCTACT field, the keyword ALIAS followed by the name (verb) of another command. The alias entry must precede the command that it references. Normally, alias entries are used in an application command table to reference system commands; for example:

| ZCTVERB | ZCTTRUNC | ZCTACT     |
|---------|----------|------------|
| QUIT    | 0        | ALIAS END  |
| EXPLAIN | 4        | ALIAS HELP |
| FORWARD | 3        | ALIAS DOWN |
| BACK    | 0        | ALIAS UP   |

This defines QUIT as an alias of END, EXPLAIN as an alias of HELP, etc. For example, when the user enters QUIT, the system responds as though END had been entered.

### Overriding System Commands

An application can override any system command simply by including the same command name in the application command table; for example:

| ZCTVERB | ZCTTRUNC | ZCTACT   |
|---------|----------|----------|
| HELP    | 0        | PASSTHRU |
| TSO     | 0        | NOP      |

In this example, the dialog has overridden both the HELP and TSO commands. During ISPF processing, if the user enters HELP, it is passed to the dialog function then in control and its function is determined by the function. The action specified for the TSO command is NOP. This disables the TSO command and if TSO is entered by a user, NOP causes the

*like "CMS" of VM ?*

command to be functionless. An "inactive command" message is automatically displayed when a NOP action command has been processed.

### Passing Commands to a Dialog Function

As previously noted, any command that is not found in the application or system command table is passed to the dialog, unaltered, in the command field. This occurs regardless of whether the command was typed in the command field or entered by use of a PF key or the attention field.

The user may force a command to be passed to the dialog, even if the command exists in the command table, by typing a greater-than sign (>) in front of the command verb.

In the command table, any command that has an action of PASSTHRU is processed as though the command were not found in the table - it is passed to the dialog in the command field.

Commands may also be passed to the dialog via the SETVERB action. This action causes the dialog manager to separate the verb from the command parameters (if any). The command verb is stored in variable ZVERB, which is in the shared variable pool. The command parameters are passed to the dialog, left-justified, in the command field; for example:

| ZCTVERB | ZCTTRUNC | ZCTACT  |
|---------|----------|---------|
| QUERY   | 0        | SETVERB |

The verb "QUERY" is stored in variable ZVERB and the character string (for example "DEPT 877") is passed in the command field.

The following user actions produce the same results:

- The user types "QUERY DEPT 877" in the command field and presses ENTER.
- The user types "DEPT 877" in the command field and then presses a PF key that has been equated to the character string "QUERY".
- The user presses a PF key that has been equated to the character string "QUERY DEPT 877".
- The user employs the light pen or cursor select key to select an attention field that contains the character string "QUERY DEPT 877".

The following system commands, distributed with the dialog manager, are defined as SETVERB action commands:

|         |       |
|---------|-------|
| END     | UP    |
| RETURN  | DOWN  |
| RFIND   | LEFT  |
| RCHANGE | RIGHT |

**Note:** The ZVERB variable can be interrogated to distinguish between END and RETURN. (The effect of END and RETURN on the DISPLAY service is the same because RETURN is used to simulate repeated END commands, until the primary option menu is reached.)

## Dynamically Specified Command Actions

A command action may be specified dynamically by means of a dialog variable. A variable action may be used to "share" commands with the dialog manager, such as UP, DOWN, LEFT, and RIGHT, and to enable or disable commands during certain points in the dialog. Suppose, for example, an application command table includes the following two entries:

| ZCTVERB | ZCTTRUNC | ZCTACT   |
|---------|----------|----------|
| UP      | 0        | &SCRVERT |
| DOWN    | 0        | &SCRVERT |

The variable SCRVERT may be used to dynamically control the action of the vertical scroll commands (UP and DOWN), as follows:

- If SCRVERT is set to NOP, the commands are disabled.
- If SCRVERT is set to PASSTHRU, the commands are passed to the dialog.
- If SCRVERT is set to blank, command scanning continues, in which case the system definitions for UP and DOWN (in the system command table) take effect.
- If SCRVERT is set to an invalid action, the commands are disabled, as in NOP.

For this particular example, setting SCRVERT to SETVERB would have the same effect as setting it to blank, because UP and DOWN are defined in the system command table as SETVERB action commands.

**Note:** If the dialog overrides or shares the use of the scroll commands, it becomes that dialog's responsibility to ensure that the commands have been redefined with an action of blank (or SETVERB) before invoking any ISPF function that requires them; namely, browse, edit, and table display. The same rule applies to the RFIND command (used by browse and edit) and the RCHANGE command (used by edit).

## TERMINAL KEYS

On the terminal, the two program access (PA) keys and the program function (PF) keys (if any) are used to request commonly used operations. No PF keys are required for ISPF operations, but ISPF is shipped with a default set of PF key definitions that users can change. Refer to Appendix B, "Using the ISPF PARMs Option," for information on specifying PF key operation.

### Program Function Keys

ISPF does not require PF keys for its operation. Commands are entered in the command field of any display (including edit, browse, member lists, and table display). PF keys are strongly recommended, however, for ease of use.

The default PF key assignments, distributed with ISPF, for the 3x4- key pad (right-hand side of the keyboard) are shown in Figure 13. These are PF keys 1-12 on a 12-key terminal, or keys 13-24 on a 24-key terminal.

For 24-key terminals, PF keys 1-12 have the same defaults as keys 13-24. It is recommended that users of 24-key terminals continue to use the key pad (13-24) for ISPF operations, and redefine PF keys 1-12 as needed by dialog applications.



|                             |                              |                     |
|-----------------------------|------------------------------|---------------------|
| PF1 / 13<br>HELP            | PF2 / 14<br>SPLIT            | PF3 / 15<br>END     |
| PF4 / 16<br>RETURN          | PF5 / 17<br>RFIND            | PF6 / 18<br>RCHANGE |
| PF7 / 19<br>SCROLL<br>UP    | PF8 / 20<br>SCROLL<br>DOWN   | PF9 / 21<br>SWAP    |
| PF10 / 22<br>SCROLL<br>LEFT | PF11 / 23<br>SCROLL<br>RIGHT | PF12 / 24<br>CURSOR |

Figure 13. Default Program Key Arrangement

### Defining PF Keys

When the KEYS command is entered or option 3 is selected from the ISPF PARMs option menu, the panel shown in Figure 14 is displayed.

The panel shown in the figure is displayed for terminals with 12 PF keys. For terminals with 24 PF keys, the first panel displayed by the KEYS command shows the "primary" keys (PF13-PF24). When the ENTER key is pressed, a panel is displayed showing the "alternate" keys (PF1-PF12). The user may flip-flop between the two panels by continuing to press ENTER.

The user may define or change a PF key function simply by equating the key to a command. Example:

```
PF9 ==> CHANGE ALL ABC XYZ
PF12 ==> PRINT
```

---

----- PF KEY DEFINITION -----

COMMAND ==> \_

NUMBER OF PF KEYS ==> 12                      TERMINAL TYPE ==> 3277

PF1 ==> HELP  
PF2 ==> SPLIT  
PF3 ==> END  
PF4 ==> RETURN  
PF5 ==> RFIND  
PF6 ==> RCHANGE  
PF7 ==> UP  
PF8 ==> DOWN  
PF9 ==> SWAP  
PF10 ==> LEFT  
PF11 ==> RIGHT  
PF12 ==> CURSOR

INSTRUCTIONS:  
Verify number of PF keys and terminal type before proceeding.  
Press ENTER key to process changes.  
Enter END command to process changes and exit.

Figure 14. PF Key Definition Panel

---

In this example, PF9 has been equated to an edit command, and PF12 has been equated to the system-defined PRINT command.

A PF key definition beginning with a colon (:) is treated as a special case. The colon is stripped off and the command to which the key is equated is inserted in the first input field on whichever line the cursor is currently positioned.

A PF key definition beginning with a greater-than sign (>) is another special case. It causes the command to be passed to the dialog regardless of whether the command appears in the command tables. This feature provides compatibility with SPF, in which edit and browse commands were defined with a greater-than sign.

**Note:** When an ISPF function is executing, do not press RESET and then attempt to enter information or use a PF key; results are unpredictable.

## Saving PF Key Definitions

PF key definitions are kept in a set of system variables named ZPF01, ZPF02, ... , ZPF24. Variables ZPF13-ZPF24 always contain the "primary" PF key definitions. For 24-key terminals, these correspond to physical keys 13-24. For 12-key terminals, these correspond to physical keys 1-12. Variables ZPF1-ZPF12 contain the "alternate" key definitions, and are meaningful only for terminals with 24 PF keys.

The current values for all 24 keys (variables ZPF01-ZPF24) are kept in the application profile. Hence, different PF key definitions can be associated with different applications.

An application can provide default PF key settings for a new user by providing a default profile. An application can prevent the user from changing the default PF key settings by overriding the KEYS command (by assigning it to NOP in the application command table).

## **MVS and VM/SP: Program Access (PA) Keys**

The two PA keys are defined as follows. These definitions may not be changed.

**ATTENTION! (PA1)** Normally, this key should not be used while you are in ISPF full screen mode. See the discussion below for exceptions.

**RESHOW (PA2)** Redisplays the contents of the screen. PA2 may be useful if a user has pressed the ERASE INPUT or CLEAR key accidentally, or has typed unwanted information but has not yet pressed ENTER or a PF key.

Generally, PA1 is used to terminate TSO commands or CLISTS running under ISPF. However, some TSO commands and CLISTS process PA1 in their own way. CLISTS with attention exists should not be run under ISPF because results are unpredictable when PA1 is pressed.

If PA1 is pressed while ISPF is in full screen mode after the keyboard has been unlocked, it is treated as a RESHOW request. If PA1 is again pressed, the current function is terminated and the primary option menu or a top-level selection panel supplied by the dialog developer, is displayed.

When an ISPF function is executing, if the RESET key is pressed to unlock the keyboard and PA1 is pressed, ISPF will attempt to terminate the current function and redisplay the primary option menu. The attempt may not always be successful (if, for example, there is an error in MVS allocation).

## VSE: Program Access (PA) Keys

The two PA keys are defined below. Because these definitions may be changed by an installation, consult the installation's system administrator or system programmer to obtain the current definitions.

**RESHOW** (PA1) Redisplays the contents of the screen. PA1 may be useful if the user has pressed the ERASE INPUT or has unwanted information but has not yet pressed ENTER or a PF key.

**CANCEL** (PA2) Normally, this key should not be used while you are in ISPF full screen mode. See the discussion below for exceptions.

The PA2 key, pressed after the keyboard has been manually unlocked (by pressing the RESET key), terminates processing and redisplay the primary option menu.

If PA2 is pressed after the keyboard has been unlocked by ISPF, it functions the same as the PA1 key. However if PA2 is pressed a second time without any intervening interaction, it terminates processing of the current function or panel and the primary option menu is redisplayed.

When an ISPF function is executing, if the RESET key is pressed to unlock the keyboard and PA2 is pressed, ISPF will attempt to terminate the current function and redisplay the primary option menu.

## LIGHT PEN AND CURSOR SELECT

ISPF permits fields on a panel to be detected with a light pen or the cursor select key. (The cursor select key is a hardware feature on 3278 and 3279 terminals.) Only the "attention" mode of light pen selection is used.

Panel fields that are to be detectable by light pen or cursor selection must be defined as attention fields. This is done with an attribute character that has been defined with the ATTN(ON) keyword. The panel designer must provide the number of blank characters before and after the attention attribute character that are required by the 3270 hardware.

Processing of light pen/cursor selected fields is handled in much the same way as PF keys. The entire contents of the selected field are treated as a command and processed as though they had been typed into the command field. If the command is found in the tables, it is executed immediately. If the command is not found in the tables, it is inserted into the command field and the entire command field is passed to the dialog.

Attention fields may be used on a menu to simulate option selection. The panel designer must truncate any unwanted characters resulting from

an attention entry into the command field. An example is shown in Figure 15.

In this example, a light pen or cursor selection of the first option would cause the character string "1 - BROWSE" to be placed in the ZCMD field and the ENTER key to be simulated. In the )PROC section, the contents of the ZCMD field are truncated at the first blank before the ZSEL variable is set based on a translation of the ZCMD field.

---

```
)ATTR
  $ TYPE(TEXT) ATTN(ON)
)BODY
%----- SOME MENU -----
%SELECT OPTION ==>_ZCMD
%
$ 1 - BROWSE +DISPLAY SOURCE DATA OR LISTINGS
$ 2 - QUERY  +FIND OUT INFORMATION ABOUT SOMETHING
.
.
.
.
)PROC
  &ZCMD = TRUNC (&ZCMD, ' ')
  &ZSEL = TRANS (TRUNC (&ZCMD, '.'))
             1, 'PGM(ISPBRO)'
             2, 'PANEL(XYZ)'
             .
             .
             .
```

Figure 15. Use of Light Pen Attribute

---

## CHAPTER 4. LIBRARY REQUIREMENTS

This chapter describes the libraries that are used by ISPF. Some are always required and others are required only if certain operations are to be performed. Libraries that are to be required during a given invocation of ISPF must be allocated before that invocation.

### MVS: LIBRARY SETUP

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

#### Required Libraries

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

| <u>DDNAME</u> | <u>DESCRIPTION</u>   | <u>RECFM</u> | <u>LRECL</u> | <u>BLKSIZE</u> |
|---------------|----------------------|--------------|--------------|----------------|
| ISPPLIB       | Panel Library        | FB           | 80           | 3120           |
| ISPMLIB       | Message Library      | FB           | 80           | 3120           |
| ISPSLIB       | Skeleton Library     | FB           | 80           | 3120           |
| ISPTLIB       | Table Input Library  | FB           | 80           | 3120           |
| ISPPROF       | User Profile Library | FB           | 80           | (see note)     |

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

A filemode number other than 1 on a minidisk other than the A-disk may not result in proper updating.

The panel, message, skeleton, and table input libraries are distributed with ISPF. There is a separate profile library for each end-user. Its contents are dynamically generated and updated during execution of ISPF.

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 used at your installation.

| <u>DDNAME</u> | <u>DSNAME</u>      |
|---------------|--------------------|
| ISPPLIB       | ISP.V1R1M0.ISPPLIB |
| ISPMLIB       | ISP.V1R1M0.ISPMLIB |
| ISPSLIB       | ISP.V1R1M0.ISPSLIB |
| ISPTLIB       | ISP.V1R1M0.ISPTLIB |
| ISPPROF       | user selected      |

Application libraries for panels, messages, skeletons, and tables should be concatenated ahead of the corresponding ISPF 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, skeletons, and tables:

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

The following allocations are required:

```
//ISPPLIB DD DSN=XYZ.PANELS,DISP=SHR
//        DD DSN=ISP.V1R1M0.ISPPLIB,DISP=SHR

//ISPMLIB DD DSN=XYZ.MSGS,DISP=SHR
//        DD DSN=ISP.V1R1M0.ISPMLIB,DISP=SHR

//ISPSLIB DD DSN=XYZ.SKELS,DISP=SHR
//        DD DSN=ISP.V1R1M0.ISPSLIB,DISP=SHR

//ISPTLIB DD DSN=ISP.V1R1M0.ISPTLIB,DISP=SHR

//ISPPROF DD DSN=USERAA.ISPF.PROFILE,DISP=OLD
```

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

## Table and File Tailoring Libraries

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

| <u>DDNAME</u> | <u>DESCRIPTION</u>    | <u>RECFM</u> | <u>LRECL</u> | <u>BLKSIZE</u> |
|---------------|-----------------------|--------------|--------------|----------------|
| ISPTABL       | Table Output Library  | FB           | 80           | (See note)     |
| ISPFIL        | 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 output library must be a partitioned data set. The ISPTABL ddname that defines it may specify the same data set as the table input library (ddname ISPTLIB) or a different data set. 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.

The table output library must be allocated to ddname ISPTABL prior to use of table services. ISPF includes ENQ logic to prevent simultaneous updates. ISPTABL must not specify a concatenated sequence of data sets.

In MVS and VM/SP, ISPTABL may be allocated dynamically by the dialog, and freed upon completion of use. In MVS, ISPTABL should be allocated with DISP=SHR even though it specifies an output data set.

**Note:** In MVS, the TSO Programming Control Facility II (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 by RACF, or allocated with DISP=OLD and protected by PCF.

File tailoring output may be written to a temporary sequential data set provided by ISPF. The temporary data set is allocated automatically, so there is no need for the dialog to allocate a data set. 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 either to a partitioned or a sequential data set. The data set must be allocated to ddname ISPFIL prior to invoking file tailoring services. ISPFIL may be allocated dynamically by the dialog, and freed upon completion. For a sequential data set, ISPFIL must be allocated with DISP=OLD. For a partitioned data set, it may be allocated with DISP=SHR, but may not be protected by the Program Control Facility II (PCF) unless it is allocated with DISP=OLD. ISPFIL 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 ISPF.

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. Alternatively, it may be in the following partitioned data set (RECFM=U):

| <u>DDNAME</u> | <u>DESCRIPTION</u> |
|---------------|--------------------|
| ISPLLIB       | ISPF 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 ISPF. 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. If both a step library and task library (ISPLLIB) are used, then the step library should be included in the ISPLLIB concatenation sequence.



**Note:** If a program is to be used in split screen mode from both screens, it should be linked as reentrant or nonreusable.

## VM/SP: LIBRARY SETUP

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

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

**Note:** Shared minidisk support is described in Appendix I, "VM/SP: Use of Shared Minidisks."

### Required Libraries

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

| <u>DDNAME</u> | <u>DESCRIPTION</u>            | <u>FILENAME</u> |
|---------------|-------------------------------|-----------------|
| ISPPLIB       | Panel Library (note 1)        | ISPPLIB MACLIB  |
| ISPMLIB       | Message Library (note 1)      | ISPMLIB MACLIB  |
| ISPSLIB       | Skeleton Library (note 1)     | ISPSLIB MACLIB  |
| ISPTLIB       | Table Input Library (note 1)  | ISPTLIB MACLIB  |
| ISPPROF       | User Profile Library (note 2) | user            |

### Notes:

1. These libraries are distributed with ISPF.
2. This library is not distributed with ISPF and is empty the first time a user logs on. There is a separate profile library for each user. Its contents are dynamically generated and updated during execution of ISPF.

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

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

```
XYZPANLS  MACLIB
XYZMSGSL  MACLIB
XYZSKELS  MACLIB
```

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

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

( FILEDEF ISPLIB DISK XYZMSGs MACLIB * (PERM CONCAT)
  FILEDEF ISPLIB DISK ISPLIB MACLIB * (PERM CONCAT)

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

**Note:** A GLOBAL MACLIB command is not required; ISPF handles the concatenation automatically based on the FILEDEF information.

These FILEDEFs must be issued prior to invoking ISPF. They may be issued in the user's PROFILE EXEC or in an EXEC that initiates the XYZ application. Any EXEC that invokes ISPF must be coded in EXEC 2 language.

**Note:** Duplicate file names are not permitted in CMS.

## Table and File Tailoring Libraries

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

| <u>DDNAME</u> | <u>Description</u>    | <u>Filename</u>      |
|---------------|-----------------------|----------------------|
| ISPTABL       | Table Output Library  | user selected MACLIB |
| ISPFIL        | File Tailoring Output | user selected MACLIB |

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 (in a FILEDEF statement) to ddname ISPTLIB prior to invoking ISPF. It may consist of a concatenated sequence of libraries, in which case the FILEDEFs must include the CONCAT parameter (see the above example). Again, a GLOBAL MACLIB command is not required.

The table output library must be allocated (using a FILEDEF statement) to ddname ISPTABL prior to use of table services. 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 ISPF. 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 a sequential file. The MACLIB or sequential file must be allocated (in a FILEDEF statement) to ddname ISPFIL before use of file tailoring services. If the MACLIB or file does not already exist, the FILEDEF statement must include a "RECFM F" parameter. The ISPFIL ddname may be allocated dynamically by the dialog, and freed (FILEDEF CLEAR) upon completion. ISPFIL must not specify a concatenated sequence of libraries.

**Note:** Table output libraries and, in some cases, file tailoring output may need to be on shared minidisks. (Shared minidisk support is further described in Appendix I, "VM/SP: Use of Shared Minidisks"). ISPF ensures the integrity of these minidisks provided all updating is done by ISPF services. However, ISPF 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 suggested:

- Isolate shared ISPF tables and file tailoring output files on minidisks that do not have other types of files.
- Caution users not to update these minidisks except through the use of ISPF services.
- Always access these minidisks as read-only extensions of themselves. This prevents inadvertent updating. For 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 prevents inadvertent updating. A FILEDEF statement for the table output library (ddname ISPTABL) is then issued to specify the particular table library (XYZTABL MACLIB) on the D-disk.

ISPF automatically reaccesses the disk, when needed, to write an updated copy of the table. ISPF then restores the original (D/D) access mode.

The same technique should be used when a table library is allocated for both input and output. For 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 coded in EXEC 2 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:

| <u>DDNAME</u> | <u>DESCRIPTION</u>            |
|---------------|-------------------------------|
| ISPXLIB       | Text Module Library (TXTLIB)  |
| ISPLLIB       | Load Module Library (LOADLIB) |

If a TXTLIB is used, it must be allocated (using a FILEDEF statement) to ddname ISPXLIB. A concatenated sequence of TXTLIBs may be specified, in which case the FILEDEF statements 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 are loaded automatically by "automatic call" reference. The called modules must also be TEXT files on an ISPF-accessible minidisk or members of the TXTLIB allocated to ddname ISPXLIB.

If a LOADLIB is used, it must be allocated (using a FILEDEF statement) 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 module references must be resolved prior to invocation by ISPF.

**Note:** Load modules may be used only for programs that are reenterable. Nested use of the same load module or concurrent use in split screen mode causes 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 by means of the SELECT parameter:

SELECT PGM(program-name)

must be relocatable (text or load module format).

In VM/SP, whenever such a program is loaded into the user area, ISPF automatically turns on CMS subset mode to prevent MODULE files from overlaying the relocatable program. ISPF 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 through use of the CONTROL service.

Dialog functions that are invoked as commands use the following SELECT parameter:

CMD(command)

This parameter may be used to invoke MODULE files if CMS is not currently operating in subset mode. However, these files may not use dialog services. If subset mode is on, any attempt to invoke or load a MODULE file will result in a CMS return code of +1.

## MVS AND VM/SP: USE OF LIBRARIES

The following steps describe the order in which MVS and VM/SP libraries are set up for use in development and test of a dialog:

1. Set up the panel, message, skeleton, table, and program libraries for the application. For the MVS environment, allocate new partitioned data sets. For the VM/SP environment, select minidisks on which the libraries are to reside, and ensure that the dialog has access to the minidisks.
2. Create a command procedure (CLIST or EXEC 2) that contains the necessary ALLOCATE or FILEDEF statements to allocate the libraries. Concatenate the application libraries ahead of the libraries required by ISPF, as described previously.
3. Create the panels, messages, and skeletons by editing directly into the application libraries. In the VM/SP environment, these libraries can be updated only in test or trace mode.
4. Create the dialog functions and assure that the text or load modules are in libraries (or on minidisks) accessible to ISPF.

**Note:** Functions coded as program modules must be link edited. Under VM/SP, they may be link edited.

In either environment, when a function is link edited, the ISPLINK subroutine must be included (explicitly or by 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/SP, ISPLINK is distributed as a TEXT file.

5. Invoke the application. To do this, add an ISPSTART command to the command procedure created in step 2. The ISPSTART command should invoke the application, using the appropriate PANEL, CMD, or PGM parameter. 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 menu.

### VSE/AF 1.3.5: LIBRARY SETUP

In VSE, source statement libraries are used to store panels, messages, tables and skeletons. The following sublibraries are used:

| <u>TYPE</u> | <u>SUBLIBRARY</u>                  |
|-------------|------------------------------------|
| MESSAGES    | M                                  |
| PANELS      | N                                  |
| SKELETONS   | S (Includes file tailoring output) |
| TABLES      | T                                  |

Source statement libraries to be used in an ISPF session are defined through ISPDEF control statements. These ISPDEF control statements are specified in the ICCF procedure used to invoke ISPF.

### Required Libraries

The following ISPDEF library definitions are required for operation of ISPF in the VSE/ICCF environment.

| <u>LIBNAME</u> | <u>DESCRIPTION</u>   |
|----------------|----------------------|
| ISPPLIB        | Panel Library        |
| ISPMLIB        | Message Library      |
| ISPSLIB        | Skeleton Library     |
| ISPTLIB        | Table Library        |
| ISPPROF        | User Profile Library |

Panel, message, skeleton, and table input libraries are provided in a source statement library distributed with ISPF. A profile library must be created for each user. Its contents are dynamically generated and updated during execution of ISPF.

Application libraries for panels, messages, skeletons, and tables should be concatenated ahead of the corresponding ISPF libraries using the libnames shown above.

### Example

Assume that application XYZ uses VSE library XYZLIB for its panels, messages, skeletons, and tables.

The following ISPDEF statements are required:

```
ISPDEF ISPPLIB,SEARCH=(XYZLIB,ISPFDM)
ISPDEF ISPLIB,SEARCH=(XYZLIB,ISPFDM)
ISPDEF ISPSLIB,SEARCH=(XYZLIB,ISPFDM)
ISPDEF ISPTLIB,SEARCH=(XYZLIB,ISPFDM)
ISPDEF ISPPROF,SEARCH=USERA,TO=USERA
```

Or, alternatively:

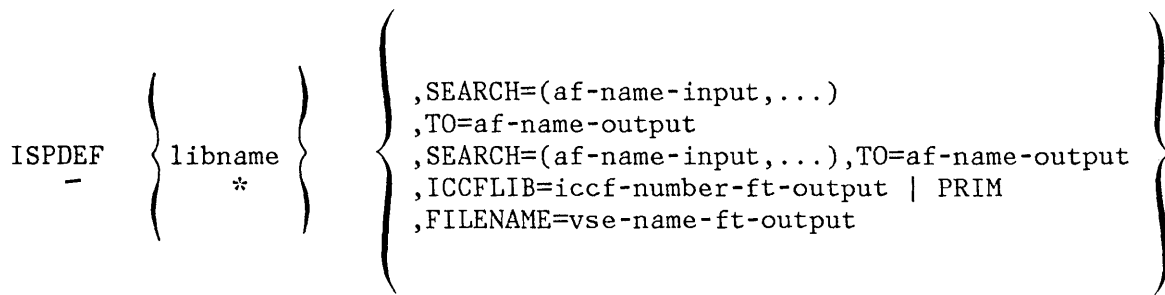
```
ISPDEF *,SEARCH=(XYZLIB,ISPFDM)
ISPDEF ISPPROF,SEARCH=USERA,TO=USERA
```

### Library Definition

The ISPDEF control statement defines libraries to be used by ISPF.

The following syntax rules apply to the ISPDEF statement:

- A statement may start in any column.
- Parameters must be separated by a comma or one or more blanks.
- A statement may be continued at any point a comma is valid and the comma must be specified.
- A continued statement may start in any column.
- Parentheses in a SEARCH parameter are optional if a single library is specified.



**libname**

Specifies the ISPF library this definition represents. The ISPF library names that may be specified are ISPLIB, ISPLIB, ISPSLIB, ISPTLIB, ISPPROF, ISPPMOD, ISPTABL and ISPFILE. User-defined names may be specified to support the LIBRARY parameter of table services and file tailoring.

**\***

Specifies that this library definition applies to all ISPF libraries that have not been explicitly defined by other library definition statements.

**af-name-input**

Specifies the name of an Advanced Function (AF) library used for input. Multiple libraries may be specified, in which case they are concatenated beginning with the initially specified library name. This parameter applies to libraries ISPLIB, ISPLIB, ISPSLIB, ISPTLIB, ISPPROF or ISPPMOD, and to the libname '\*'. A maximum of 15 libraries may be specified.

**af-name-output**

Specifies the name of an AF library used for output. This parameter applies to libraries ISPPROF, ISPPMOD, ISPTABL, ISPFILE, user defined libraries, and to the libname '\*'.

**iccf-number-ft-output**

Specifies the number of an ICCF library number to be used for file tailoring output. This parameter applies to library ISPFILE and to user defined libraries.

**vse-name-ft-output**

Specifies the filename of a VSE sequential file to be used for file tailoring output. This parameter applies to library ISPFILE only.

Figure 16 shows the ISPDEF statement parameters and the ISPF libraries that may be specified in each. Figure 17 shows relationships between defaults specified by the '\*' libname statement and unspecified ISPF libraries.



| libname         |   | SEARCH | TO | ICCFLIB | FILENAME |
|-----------------|---|--------|----|---------|----------|
| ISPPLIB         | M | R      | NA | NA      | NA       |
| ISPMLIB         | M | R      | NA | NA      | NA       |
| ISPSLIB         | M | R      | NA | NA      | NA       |
| ISPTLIB         | M | R      | NA | NA      | NA       |
| ISPPROF         | M | R1     | R1 | NA      | NA       |
| ISPPMOD         | O | R1     | R1 | NA      | NA       |
| ISPTABL         | O | NA     | R  | NA      | NA       |
| ISPFIL          | O | NA     | R2 | R2      | R2       |
| user<br>defined | O | NA     | R2 | R2      | NA       |
| *               | O | R3     | R3 | NA      | NA       |

Figure 16. ISPDEF Statement Parameters and Libraries to Which They Apply

Meanings for the codes used in Figure 16 are:

- M** - Mandatory ISPF library
- NA** - Not applicable
- O** - Optional ISPF library
- R** - Required parameter
- R1** - The library specified for ISPPROF and ISPPMOD is used for input and output operations. A single library must be specified using either the SEARCH or TO parameters. The same library must be indicated if both the SEARCH and TO parameters are specified.
- R2** - A single AF library or ICCF library or VSE sequential data set filename must be specified. For a user defined library used by table services, only the TO parameter is valid.

R3 - The libname '\*' specifies the default input library chain and/or default output library. The SEARCH parameter specifies one or more libraries as the default ISPF input library chain. The TO parameter specifies one library as the default ISPF output library. Refer to Figure 17 for the relationship between the '\*' libname statement and unspecified ISPF libraries.

| libname            | SOURCE OF DEFAULT LIBRARY SPECIFICATION                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ISPPLIB            | libraries specified by the SEARCH parameter                                                                                                                                              |
| ISPM LIB           | libraries specified by the SEARCH parameter                                                                                                                                              |
| ISPSLIB            | libraries specified by the SEARCH parameter                                                                                                                                              |
| ISPTLIB            | libraries specified by the SEARCH parameter                                                                                                                                              |
| ISPPROF<br>ISPPMOD | first library specified by the SEARCH parameter or library specified by the TO parameter. The same default library must result if both the SEARCH and TO defaults have been established. |
| ISPTABL            | library specified by the TO parameter                                                                                                                                                    |
| ISPFIL E           | library specified by the TO parameter                                                                                                                                                    |

Figure 17. Relationship Between Defaults Specified by the '\*' Libname Statement and Unspecified ISPF Libraries.

Three examples of ISPDEF statements are given below. In example 1, all ISPF input libraries are defined by USER1 followed by the ISPF product source library, and all ISPF output libraries are defined to USER1. Example 2 defines the same ISPF libraries as example 1, but uses the '\*' libname.

Example 3 explicitly specifies an ISPPROF definition. In this case, if ISPPROF were not explicitly specified, the default definition from the '\*' ISPDEF statement would be in effect and would be invalid. The default is invalid because the first library in the SEARCH parameter is not the same as the library in the TO parameter. In this example, the optional ISPF library ISPPMOD will not be defined from the defaults established because the result also would be invalid.

**Note:** The name ISPFDM S, used in these examples, is the recommended name.

1. ISPDEF ISPPLIB,SEARCH=(USER1,ISPFDMDS)  
 ISPDEF ISPMLIB,SEARCH=(USER1,ISPFDMDS)  
 ISPDEF ISPSLIB,SEARCH=(USER1,ISPFDMDS)  
 ISPDEF ISPTLIB,SEARCH=(USER1,ISPFDMDS)  
 ISPDEF ISPPROF,SEARCH=USER1,TO=USER1  
 ISPDEF ISPPMOD,SEARCH=USER1,TO=USER1  
 ISPDEF ISPTABL,TO=USER1  
 ISPDEF ISPFILE,TO=USER1  
 ISPDEF USERDEF,ICCFLIB=4
2. ISPDEF \*,SEARCH=(USER1,ISPFDMDS),TO=USER1  
 ISPDEF USERDEF,ICCFLIB=4
3. ISPDEF \*,SEARCH=(USER2,USER1,ISPFDMDS),TO=USER1  
 ISPDEF ISPPROF SEARCH=USER1,TO=USER1  
 ISPDEF USERDEF,ICCFLIB=4

### Table and File Tailoring Libraries

The following libraries are optional, and need be specified only when an application is to use table or file tailoring services.

| <u>LIBNAME</u> | <u>DESCRIPTION</u>    |
|----------------|-----------------------|
| ISPTABL        | Table Output Library  |
| ISPFILE        | File Tailoring Output |

The table output library must be a VSE library. The ISPTABL definition may specify the same library as the table input library or a different library. The libraries must be the same if the updated version of a table is to be reprocessed by the same dialog that updated it.

File tailoring output may be written to a temporary sequential data set defined under the filename "ISPCTLn". The filename of the temporary data set is available in system variable ZTEMPF. (When accessing this data set use RECFM of fixed, BLKSIZE of 800, and LRECL of 80.)

If the temporary data set is not used, file tailoring output may be written to a VSE library, an ICCF library, or a sequential data set as specified by the ISPDEF statement for ISPFILE. (When accessing this data set use fixed RECFM, BLKSIZE of 800, and LRECL of 80.)

### VSE/AF 1.3.5: USE OF LIBRARIES

The following steps describe the order in which VSE libraries are set up for use in development and test of a dialog:

1. Allocate an AF private source statement library to contain the panels, message, skeletons, and tables associated with the dialog.
2. Create an ICCF procedure that contains the necessary ISPDEF statements to define the libraries to be used during ISPF execution.

The sample ISPSTART ICCF procedure located in the ICCF common library should be used as a guide. (This procedure is placed in the ICCF common library during ISPF installation.)

3. Create the panels, messages, and skeletons by editing directly into the application library (allocated in step 1).
4. Create the dialog functions and make them available in core image libraries accessible to ISPF. Each dialog function must be link edited and the subroutine ISPLINK must be included (explicitly or by autolink) in the phase.
5. Invoke the application. To do this, modify the ISPSTART command in the ICCF procedure created in step 2. The ISPSTART command should invoke the application using the appropriate panel or pgm parameter. This ICCF procedure may be made available to application users as a means of invoking the application. Alternatives are to invoke the application from the master menu or other menu.

## VSE/AF 2.1: LIBRARY SETUP

In VSE, libraries.sublibraries are used to store panels, messages, tables and skeletons. The following member types are used:

| <u>CATEGORY</u> | <u>TYPE</u> |                                  |
|-----------------|-------------|----------------------------------|
| MESSAGES        | M           |                                  |
| PANELS          | N           |                                  |
| SKELETONS       | S           | (Includes file tailoring output) |
| TABLES          | T           |                                  |

Libraries.sublibraries to be used in an ISPF session are defined through ISPDEF control statements. These ISPDEF control statements are specified in the ICCF procedure used to invoke ISPF.

### Required Libraries

The following ISPDEF library definitions are required for operation of ISPF in the VSE/ICCF environment.

| <u>LIBNAME</u> | <u>DESCRIPTION</u>   |
|----------------|----------------------|
| ISPPLIB        | Panel Library        |
| ISPMLIB        | Message Library      |
| ISPSLIB        | Skeleton Library     |
| ISPTLIB        | Table Library        |
| ISPPROF        | User Profile Library |

Panel, message, skeleton, and table input libraries are provided in a VSE/AF library.sublibrary distributed with ISPF. A profile library.sublibrary must be created for each user. Its contents are dynamically generated and updated during execution of ISPF.

Application libraries.sublibraries for panels, messages, skeletons, and tables should be concatenated ahead of the corresponding ISPF libraries using the libnames shown above.

### Example

Assume that application XYZ uses VSE library.sublibrary XYZLIB.XYZSUB for its panels, messages, skeletons, and tables.

The following ISPDEF statements are required:

```
ISPDEF ISPPLIB,SEARCH=(XYZLIB.XYZSUB,ISPF.DM)
ISPDEF ISPMLIB,SEARCH=(XYZLIB.XYZSUB,ISPF.DM)
ISPDEF ISPSLIB,SEARCH=(XYZLIB.XYZSUB,ISPF.DM)
ISPDEF ISPTLIB,SEARCH=(XYZLIB.XYZSUB,ISPF.DM)
ISPDEF ISPPROF,SEARCH=USERLIB.USERA,TO=USERLIB.USERA
```

Or, alternatively:

```
ISPDEF *,SEARCH=(XYZLIB.XYZSUB,ISPF.DM)
ISPDEF ISPPROF,SEARCH=USERLIB.USERA,TO=USERLIB.USERA
```

### Library Definition

The ISPDEF control statement defines VSE libraries.sublibraries, ICCF libraries, or VSE sequential files to be used by ISPF.

The following syntax rules apply to the ISPDEF statement:

- A statement may start in any column.
- Parameters must be separated by a comma or one or more blanks.
- A statement may be continued at any point a comma is valid and the comma must be specified.
- A continued statement may start in any column.
- Parentheses in a SEARCH parameter are optional if a single library.sublibrary is specified.

---

|        |                                                                    |                                                                                                                                                                                                                           |
|--------|--------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ISPDEF | $\left. \begin{array}{c} \text{libname} \\ * \end{array} \right\}$ | $\left( \begin{array}{l} ,SEARCH=(af-name-input,...) \\ ,TO=af-name-output \\ ,SEARCH=(af-name-input,...),TO=af-name-output \\ ,ICCFLIB=iccf-number-ft-output   PRIM \\ ,FILENAME=vse-name-ft-output \end{array} \right)$ |
|--------|--------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

**libname**

Specifies the ISPF library this definition represents. The ISPF library names that may be specified are ISPPLIB, ISPMLIB, ISPSLIB, ISPTLIB, ISPPROF, ISPPMOD, ISPTABL and ISPFIL. User-defined names may be specified to support the LIBRARY parameter of table services and file tailoring.

**\***

Specifies that this library definition applies to all ISPF libraries that have not been explicitly defined by other library definition statements.

**af-name-input**

Specifies the name of a VSE/Advanced Function (AF) library.sublibrary used for input. Multiple libraries.sublibraries may be specified, in which case they are concatenated beginning with the initially specified library.sublibrary name. This parameter applies to libraries ISPPLIB, ISPMLIB, ISPSLIB, ISPTLIB, ISPPROF or ISPPMOD, and to the libname '\*'. A maximum of 15 libraries.sublibraries may be specified.

**af-name-output**

Specifies the name of a VSE/AF library.sublibrary used for output. This parameter applies to libraries ISPPROF, ISPPMOD, ISPTABL, ISPFIL, user defined libraries, and to the libname '\*'.

**iccf-number-ft-output**

Specifies the number of an ICCF library number to be used for file tailoring output. This parameter applies to library ISPFIL and to user defined libraries.

**vse-name-ft-output**

Specifies the filename of a VSE sequential file to be used for file tailoring output. This parameter applies to library ISPFIL only.

Figure 18 shows the ISPDEF statement parameters and the ISPF libraries that may be specified in each. Figure 19 shows relationships between

defaults specified by the '\*' libname statement and unspecified ISPF libraries.

| libname      |   | SEARCH | TO | ICCFLIB | FILENAME |
|--------------|---|--------|----|---------|----------|
| ISPPLIB      | M | R      | NA | NA      | NA       |
| ISPMLIB      | M | R      | NA | NA      | NA       |
| ISPSLIB      | M | R      | NA | NA      | NA       |
| ISPTLIB      | M | R      | NA | NA      | NA       |
| ISPPROF      | M | R1     | R1 | NA      | NA       |
| ISPPMOD      | O | R1     | R1 | NA      | NA       |
| ISPTABL      | O | NA     | R  | NA      | NA       |
| ISPFIL       | O | NA     | R2 | R2      | R2       |
| user defined | O | NA     | R2 | R2      | NA       |
| *            | O | R3     | R3 | NA      | NA       |

Figure 18. ISPDEF Statement Parameters and Libraries to Which They Apply

Meanings for the codes used in Figure 18 are:

- M - Mandatory ISPF library
- NA - Not applicable
- O - Optional ISPF library
- R - Required parameter
- R1 - The library.sublibrary specified for ISPPROF and ISPPMOD is used for input and output operations. A single library.sublibrary must be specified using either the SEARCH or TO parameters. The same library.sublibrary must be indicated if both the SEARCH and TO parameters are specified.

- R2 - A single VSE/AF library.sublibrary or ICCF library or VSE sequential data set filename must be specified. For a user defined library used by table services, only the TO parameter is valid.
- R3 - The libname '\*' specifies the default input library chain and/or default output library. The SEARCH parameter specifies one or more libraries.sublibraries as the default ISPF input library chain. The TO parameter specifies one library.sublibrary as the default ISPF output library. Refer to Figure 19 for the relationship between the '\*' libname statement and unspecified ISPF libraries.

| libname            | SOURCE OF DEFAULT LIBRARY.SUBLIBRARY SPECIFICATION                                                                                                                                                                        |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ISPPLIB            | libraries.sublibraries specified by the SEARCH parameter                                                                                                                                                                  |
| ISPM LIB           | libraries.sublibraries specified by the SEARCH parameter                                                                                                                                                                  |
| ISPSLIB            | libraries.sublibraries specified by the SEARCH parameter                                                                                                                                                                  |
| ISPTLIB            | libraries.sublibraries specified by the SEARCH parameter                                                                                                                                                                  |
| ISPPROF<br>ISPPMOD | first library.sublibrary specified by the SEARCH parameter or library.sublibrary specified by the TO parameter. The same default library.sublibrary must result if both the SEARCH and TO defaults have been established. |
| ISPTABL            | library.sublibrary specified by the TO parameter                                                                                                                                                                          |
| ISPFIL E           | library.sublibrary specified by the TO parameter                                                                                                                                                                          |

Figure 19. Relationship Between Defaults Specified by the '\*' Libname Statement and Unspecified ISPF Libraries.Sublibraries

Three examples of ISPDEF statements are given below. In example 1, all ISPF input libraries are defined by USERLIB.USER1 followed by the ISPF product library, and all ISPF output libraries are defined to USERLIB.USER1. Example 2 defines the same ISPF libraries as example 1, but uses the '\*' libname.

Example 3 explicitly specifies an ISPPROF definition. In this case, if ISPPROF were not explicitly specified, the default definition from the '\*' ISPDEF statement would be in effect and would be invalid. The default is invalid because the first library in the SEARCH parameter is not the same as the library in the TO parameter. In this example, the



optional ISPF library ISPPMOD will not be defined from the defaults established because the result also would be invalid.

**Note:** The name ISPF.DM, used in these examples, is the recommended name.

1. ISPDEF ISPLLIB,  
SEARCH=(USERLIB.USER1,  
ISPF.DM)  
ISPDEF ISPLLIB,SEARCH=(USERLIB.USER1,ISPF.DM)  
ISPDEF ISPSLIB,SEARCH=(USERLIB.USER1,ISPF.DM)  
ISPDEF ISPTLIB,SEARCH=(USERLIB.USER1,ISPF.DM)  
ISPDEF ISPPROF,SEARCH=USERLIB.USER1,TO=USERLIB.USER1  
ISPDEF ISPPMOD,SEARCH=USERLIB.USER1,TO=USERLIB.USER1  
ISPDEF ISPTABL,TO=USERLIB.USER1  
ISPDEF ISPFILE,TO=USERLIB.USER1  
ISPDEF USERDEF,ICCFLIB=4
2. ISPDEF \*,SEARCH=(USERLIB.USER1,ISPF.DM),TO=USERLIB.USER1  
ISPDEF USERDEF,ICCFLIB=4
3. ISPDEF \*,  
SEARCH=(USERLIB.USER2,USERLIB.USER1,ISPF.DM),  
TO=USERLIB.USER1  
ISPDEF ISPPROF SEARCH=USERLIB.USER1,TO=USERLIB.USER1  
ISPDEF USERDEF,ICCFLIB=4

## Table and File Tailoring Libraries

The following libraries are optional, and need be specified only when an application is to use table or file tailoring services.

| <u>LIBNAME</u> | <u>DESCRIPTION</u>    |
|----------------|-----------------------|
| ISPTABL        | Table Output Library  |
| ISPFILE        | File Tailoring Output |

The table output library must be a VSE library.sublibrary. The ISPTABL definition may specify the same library.sublibrary as the table input library or a different library.sublibrary. The library.sublibraries must be the same if the updated version of a table is to be reprocessed by the same dialog that updated it.

File tailoring output may be written to a temporary sequential data set defined under the filename "ISPCTLn". The filename of the temporary data set is available in system variable ZTEMPF. (When accessing this data set use RECFM of fixed, BLKSIZE of 800, and LRECL of 80.)

If the temporary data set is not used, file tailoring output may be written to a VSE library.sublibrary, an ICCF library, or a sequential data set as specified by the ISPDEF statement for ISPFILE. (When accessing this data set use fixed RECFM, BLKSIZE of 800, and LRECL of 80.)

## VSE/AF 2.1: USE OF LIBRARIES

The following steps describe the order in which VSE/AF libraries.sublibraries are set up for use in development and test of a dialog:

1. Define an AF library.sublibrary to contain the panels, message, skeletons, and tables associated with the dialog.
2. Create an ICCF procedure that contains the necessary ISPDEF statements to define the libraries to be used during ISPF execution. The sample ISPSTART ICCF procedure located in the ICCF common library should be used as a guide. (This procedure is placed in the ICCF common library during ISPF installation.)
3. Create the panels, messages, and skeletons by editing directly into the application library.sublibrary (defined in step 1).
4. Create the dialog functions and make them available in libraries.sublibraries accessible to ISPF. Each dialog function must be link edited and the subroutine ISPLINK must be included (explicitly or by autolink) in the phase.
5. Invoke the application. To do this, modify the ISPSTART command in the ICCF procedure created in step 2. The ISPSTART command should invoke the application using the appropriate panel or pgm parameter. This ICCF procedure may be made available to application users as a means of invoking the application. Alternatives are to invoke the application from the master menu or other menu.



## CHAPTER 5. INVOCATION AND TERMINATION

This chapter describes how to invoke ISPF in both interactive and batch environments, and how to terminate ISPF processing.

### MVS AND VM/SP: INVOCATION OF ISPF

In MVS and VM/SP, ISPF is invoked using the ISPSTART command. The ISPSTART command may be issued:

- By the user at a terminal
- From a command procedure (CLIST or EXEC 2)
- During LOGON (from a TSO LOGON procedure or CMS PROFILE EXEC)

When PDF is installed, the ISPF command may be used to invoke either ISPF/PDF or other dialogs if a PANEL, CMD, or PGM keyword is specified. The ISPF command provides compatibility with the SPF Program Product.

### VSE: INVOCATION OF ISPF

In VSE, ISPF is invoked by an ICCF procedure. The procedure may have any name acceptable to ICCF, however, in this publication, ISPSTART is the name assumed. The procedure to invoke ISPF consists of the following kinds of statements:

```
&&OPTIONS 0010001
/INPUT
&/LOAD ISPSTART
&/OPTION GETVIS P-22 TIME=32767,65535
&/FILE ISPLOG,DISP=DELETE,SPACE=2
&/FILE ISPLIST,DISP=DELETE,SPACE=2
&/FILE ISPCTL1,DISP=DELETE,SPACE=2
&/FILE ISPCTL2,DISP=DELETE,SPACE=2
ISPSTART &&PARAM1 &&PARAM2 &&PARAM3 &&PARAM4
ISPDEF *,SEARCH=(...,ISPFDM),TO=...      (for VSE/AF 1.3.5)
ISPDEF *,SEARCH=(...,ISPF.DM),TO=...     (for VSE/AF 2.1)
/END
/PEND
/RUN
```

The /LOAD job entry statement invokes ISPF.

The /OPTION job entry statement specifies a maximum GETVIS area size.

The /FILE job entry statements define the log, list, and temporary data sets to be used during ISPF processing.

ISPDEF command processing is described in Chapter 4, "Library Requirements."

*" is an EXEC" within VM*

## ISPSTART SYNTAX

This section describes syntax for the command that invokes ISPF. In MVS and VM/SP, this command is the ISPSTART command. In VSE, the command invokes an ICCF procedure that, in turn, invokes ISPF. Generally, this ICCF procedure is named ISPSTART. Other names may be used; however, the name ISPSTART is used in this publication.

Notation conventions are described in the Preface.

In MVS and VM/SP, the command and its parameters are coded as shown below.

In VSE, command parameters are coded as shown below, but with the parameter string enclosed in apostrophes when any parameter in the string includes parenthesis, as in the following example.

```
ISPSTART 'PANEL(USER) NEWAPPL(ZZZZ)'
```

The resulting ISPSTART command statement, after ICCF substitution, is:

```
ISPSTART PANEL(USER) NEWAPPL(ZZZZ)
```

*similar to ISPDCS within VM*

The format for ISPSTART is:

---

```
ISPSTART { PANEL(panel-name) [OPT(option)]
          }
          { CMD(command)
          }
          { PGM(program-name) [PARM(parameters)]
            [LANG(PLI|PL1 [,storage-area])]
            [LANG(COBOL)]
          }
          [NEWAPPL[(application-id)]]
          [TEST|TESTX|TRACE|TRACEX]
```

---

**panel-name**

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

**option**

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

**command**

In MVS and VM/SP, specifies a command procedure (CLIST or EXEC 2) 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 EXEC 2 name to improve performance.

In VSE, this parameter is not used because ICCF procedures may not be used for writing dialog functions.

**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.

**Note:** Dialog developers should avoid using the ISP and ISR prefixes (the ISPF and PDF component codes) in naming dialog functions. Special linkage conventions, intended only for internal ISPF use, are used to invoke programs named "ISPxxxxx" and "ISRxxxxx".

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

In VM/SP, 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/SP Environment" for more information.

(see p 08)

### 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 the same as that for passing parameters by use of the PARM= keyword on a JCL EXEC statement.

Parameters on the ISPSTART command to be passed to a PL/I program are coded 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 ISPF dialog variable, it must be assigned to a fixed character string because the VDEFINE service cannot handle varying length PL/I strings.

In MVS and VM/SP, 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.

### LANG(PLI) or LANG(PL1)

In VSE, specifies that the function being invoked is written in the PL/I language.

In MVS and VM/SP, this keyword is not used.

### storage-area

In VSE, for programs written in PL/I, specifies the number of bytes of dynamic storage to be made available to the function being invoked.

The default storage size is 2816 bytes. Generally, the amount of storage required for a PL/I program can be determined by using the storage option to compile the program and then adding 2192 to the DSA sizes specified on the compile listing. However, the storage requirement will vary considerably depending on the compiler options specified, i.e. "FLOW" will use slightly more storage but "COUNT" will substantially increase the storage requirement.

In MVS and VM/SP, this parameter is not used.

### LANG(COBOL)

In VSE, specifies that the function being invoked is written in the COBOL language.

In MVS and VM/SP, this keyword is not used.

### NEWAPPL(application-id)

Specifies a 1- to 4-character code that identifies the application that is being invoked. The code is to be prefixed to the user and edit profile names or the command table associated with the application, as follows:

```
User Profile   - xxxxPROF
Edit Profile   - xxxxEDIT
Command Table  - xxxxCMDS
```

where xxxx is the application-id. If the application-id is omitted, or if the NEWAPPL keyword is omitted, the application-id defaults to ISP.

### TEST

Specifies that ISPF is to be operated in TEST mode, described below.

### TESTX

Specifies that ISPF is to be operated in extended TEST mode, described below.

### TRACE

Specifies that ISPF is to be operated in TRACE mode, described below.

### TRACEX

Specifies that ISPF is to be operated in extended TRACE mode, described below.

The return code from ISPSTART is always 0.

## Test Modes

The testing modes of ISPF provide special processing actions to help in the debugging of a dialog. If PDF is installed, consider using the dialog test option of that facility (see [ISPF/PDF Reference](#)), instead of the testing modes described here.

Any one of four mutually exclusive keyword parameters may be specified on the ISPSTART command to control the operational mode when testing a dialog:

- TEST - Test mode
- TESTX - Extended test mode
- TRACE - Trace mode
- TRACEX - Extended trace mode



In TEST mode, ISPF 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 ISPF service. (In normal mode, the most recently accessed panel definitions are retained in virtual storage and, under MVS, a table of TTRs returned from BLDL macros is kept in virtual storage for frequently used message, panel, skeleton, and table members. 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.

Under MVS, a new extent on a DASD may be caused when using an editor to modify a panel, message, or skeleton or by link editing a module. When a new extent is allocated, the modification can be accessed only by first terminating and then reinvoking ISPF.

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 through use of the PRINT or PRINT-HI commands) include line numbers, current panel name, and message id.
4. If a dialog function is operating in the CANCEL error mode (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, however, are unpredictable and may result in an ABEND.
5. Other than the situation described in item 4, any ISPF-detected error, ABEND, or program interrupt forces an ABEND of ISPF. The user may also force an ABEND by entering ABEND or CRASH in the command line of any panel.
6. For MVS/TSO:

The PA1 key causes an immediate exit from ISPF.

If an ISPF subtask ABENDs, a dump may be taken by pressing ENTER after the ABEND message appears, provided that a SYSUDUMP, SYSMDUMP, or SYSABEND data set has been allocated.

7. For VM/CMS:

An ADSTOP set within ISPF code is not lost, even if ISPF invokes a CMS command that executes in the user area. If ISPF is operating in DCSS, the page containing the ADSTOP is marked non-shareable, and is copied automatically to the user area.

In TESTX (extended test) mode, ISPF operates the same as in TEST mode except that all messages written to the ISPF log file are also displayed at the terminal.

## Trace Modes

In TRACE mode, ISPF operates the same as it does in TEST mode, with the following exception.

In MVS, VM/SP, and ISPF/PDF option 7.6, a message is written to the ISPF log file whenever any ISPF service is invoked (even if CONTROL ERRORS RETURN has been issued) and whenever any error is detected by an ISPF service. Note that only CLIST and EXEC 2 service requests, and service requests issued under PDF option 7.6 are recorded; program module requests for service are not recorded in the log file.

In TRACEX (extended trace) mode, ISPF operates the same as it does in TRACE mode except that all messages written to the ISPF log file (including the trace messages) are also displayed at the terminal.

## DIALOG INITIATION AND TERMINATION

Execution of a dialog is initiated by the SELECT service. Selection keywords, passed to the SELECT service, specify whether the dialog begins with the display of a menu (PANEL keyword) or the execution of a dialog function (CMD or PGM keyword). The dialog terminates when the selected menu or function terminates. The action at termination depends upon how the SELECT service was originally invoked.

## SELECT Service Invocation

The SELECT service may be invoked in the following ways:

- During initialization, the SELECT service is automatically invoked by the dialog manager to initiate the first dialog. The selection keywords originally specified on the ISPSTART command are passed to the SELECT service.

For dialogs invoked by ISPSTART, ISPF error processing is not put into effect until ISPF initialization is completed.

- If the user enters split screen mode, the dialog manager again invokes the SELECT service and again passes the selection keywords from the ISPSTART command. This causes the first dialog (specified in the ISPSTART command) to be initiated on the second logical screen.

In VSE, user dialog functions are restricted to one logical screen. That is, a user dialog function may be executed in either logical screen, but may not be executed in both logical screens

concurrently. This restriction does not apply to ISPF or ISPF/PDF functions.

- The SELECT service recursively invokes itself whenever the user selects an option from a menu displayed by the SELECT service. In this case, the selection keywords are specified in the panel definition for the menu.
- The SELECT service may be invoked from a dialog function. In this case, the selection keywords are passed as calling sequence parameters.

The action taken at dialog termination is as follows:

- If the SELECT service was invoked from a dialog function, control is returned to that function and the function continues execution.
- If the SELECT service was invoked from a menu, that menu is redisplayed (including execution of the INIT section in the panel definition).
- If the user is terminating split screen mode, the original dialog is ended on that logical screen and the other logical screen expands to the full size of the physical display screen.
- If the user is terminating ISPF (which can only be done in single screen mode), either the ISPF termination panel is displayed or the user's defaults for list/log processing are employed (as specified using the ISPF PARMs option).

The termination panel is displayed if:

- The dialog started with the display of a menu and the user entered the END command on that menu.
- The dialog started with the execution of a function and the function ended with a return code of 0.

The list/log defaults are used if:

- The dialog started with the display of a menu and the user entered the RETURN command or selected the "exit" option (see "Special Panel Requirements" in Chapter 7 for discussion of the EXIT keyword).
- The dialog started with the execution of a function and the function ended with a return code of 4 or higher. (A return code higher than 4 will cause an error message to be displayed.)

If the user has not specified valid list/log defaults, the ISPF termination panel is displayed in all cases.

## VSE: Dialog Abend Intercept

In VSE, the STXIT AB facilities of VSE are available to dialogs. Each function of a dialog - invoked using SELECT PGM(program-name) - may issue the STXIT AB macro to establish its own STXIT AB exit.

ISPF manages dialog STXIT AB exit information so that each function of the dialog operates independently. If an abnormal condition is detected, each dialog STXIT AB exit routine that is active is given control prior to ISPF terminating the currently running logical screen. This permits data needed for debugging to be saved before the dialog is terminated.

Because ISPF and user dialogs operate in the same partition, an error that causes a dialog to abend may also destroy ISPF's ability to process dialog service requests. Therefore, requests for dialog services included within an STXIT AB exit may themselves cause an abend and should be avoided or, at least, not specified until all other dialog recovery processing has been specified.

**Note:** When a dialog is cancelled by the system operator or because execution time expires, dialog STXITs do not receive control.

## BATCH EXECUTION OF ISPF SERVICES

When initiated in a batch environment, ISPF services execute as a command in the background. Only services that are non-interactive execute successfully. Any services that cause a full screen write result in an error message. Background invocations are generally used to invoke ISPF table and file tailoring services; however, access to other non-interactive dialog services are also available.

## TSO Batch Environment

TSO provides facilities for executing command processors in the batch environment. The JCL stream provides for data sets to be pre-allocated prior to the invocation of any command. The Terminal Monitor Program (TMP) is invoked by use of the EXEC statement, and establishes the necessary control blocks for the TSO environment. The command input stream is accessed from the SYSTSIN DD statement and all terminal line I/O outputs issued by the TSO I/O service routines are directed to the SYSTSPRT DD statement definition. The ISPF libraries are allocated using DD statements. The panels, messages, skeleton, table, and profile data sets must be preallocated. While not required, it is recommended that the log data set also be preallocated. If a log data set is dynamically allocated, it is always kept at ISPF termination.

The ISPF command is placed in the input stream with the CMD or PGM keywords that name the dialog to be invoked. All dialog services are permitted except for BROWSE, DISPLAY, EDIT, SELECT PANEL, SETMSG, and TBDISPL.

A userid is selected for the background job, as follows:

1. If available, the userid supplied during RACF authorization checking is used.
2. If a userid is not available from RACF, the prefix supplied with the TSO PROFILE command is used.
3. If neither of the above occurs, the default is "BATCH."

### Sample Batch Job

Figure 20 shows a sample batch job. This job invokes the MVS/TSO Terminal Monitor Program (TMP) which, in MVS, establishes the environment necessary to attach command processors. The ISPSTART command is specified in the TSO background input stream (SYSTSIN) with the name of a CLIST (TBUPDATE) that contains the ISPF services to be executed.

### Error Processing

ISPF terminates with an error message if a required library is not available. The ISPSTART command must also be invoked naming either a CLIST or PGM function. If no dialog is specified, a message is issued. These messages are directed to the file defined by the SYSTSPRT DD statement.

---

```

//USERAA JOB (AAO4,BIN1,000000),'I. M. USERAA',
// CLASS=L,MSGCLASS=A,NOTIFY=USERAA,MSGLEVEL=(1,1)
//*-----*/
//* EXECUTE ISPF COMMAND IN THE BACKGROUND */
//*-----*/
//ISPFBACK EXEC PGM=IKJEFT01,DYNAMNBR=25,REGION=1024K
//*
//*- - - ALLOCATE PROFILE, PANELS, MSGS, PROCS, AND LOG - -
//ISPPROF DD DSN=USERAA.ISPF.PROFILE,DISP=OLD
//ISPPLIB DD DSN=ISP.V1R1M0.ISPPLIB,DISP=SHR
//ISPMLIB DD DSN=ISP.V1R1M0.ISPMLIB,DISP=SHR
//ISPSLIB DD DSN=ISP.V1R1M0.ISPSLIB,DISP=SHR
//ISPLOG DD DSN=USERAA.ISPF.LOG,DISP=SHR
//*
//*- - - ALLOCATE TABLE DATA SETS - - - - - */
//ISPTLIB DD DSN=ISP.V1R1M0.ISPTLIB,DISP=SHR
//ISPTABL DD DSN=USERAA.ISPF.TABLES,DISP=SHR
//*
//*- - - ALLOCATE DIALOG PROGRAM AND CLIST LIBRARIES- - - */
//ISPLLIB DD DSN=USERAA.ISPF.LOAD,DISP=SHR
//SYSPROC DD DSN=USERAA.ISPF.CLIST,DISP=SHR
//*
//*- - - ALLOCATE TSO BACKGROUND OUTPUT AND INPUT DS- - - */
//SYSTSPRT DD DSN=USERAA.ISPF.ISPFPRNT,DISP=SHR
//SYSTSIN DD *
    PROFILE PREFIX(USERAA)          /* ESTABLISH PREFIX      */
    ISPSTART CMD(%TBUPDATE)         /* INVOKE CLIST DIALOG  */
/*

```

Figure 20. MVS Batch Job

---

Errors encountered during background dialog execution are handled in the same manner as errors encountered during foreground execution. Messages normally written to the ISPF log data set for severe errors are also written to the SYSTSPRT file. This is useful when executing a CLIST dialog because any error messages are listed immediately after the ISPEXEC service in which the error occurred.

If a function encounters an abend, the entire ISPF batch job stream terminates. A message is issued to the SYSTSPRT file indicating the type of abend.

## VM/SP Batch Environment

A disconnected virtual machine or a CMS batch machine can be used to execute non-interactive dialogs. The command inputs may be specified via the CMS Console stack or by using the CMS PUNCH command to provide input to the batch machine reader. In either case, the ISPF libraries must be specified using FILEDEF commands for the panels, messages, skeleton, tables, and profile maclibs.

All dialog services may be invoked except BROWSE, DISPLAY, EDIT, SELECT PANEL, SETMSG, and TBDISPL.

### Sample Batch Job

Figure 21 shows a sample batch job.

This job provides the links and accesses needed by the batch machine to invoke ISPF with the correct libraries. This EXEC also sends any list and log files back to the originator. The profile tables could also be sent.

### Error Processing

ISPF terminates with an error message if a required library is not available. The ISPSTART command must also be invoked naming either an EXEC or a PGM function. If no dialog is specified, a message is issued. These messages are directed to the console log.

Errors encountered during background execution are handled in the same manner as errors encountered during normal execution. Messages normally written to the ISPF log file for severe errors are also written to the CMS Console Log.

---

```

* * * * *
* BUILD THE BATCH JOB CARDS *
* * * * *

CP SPOOL PUNCH CLASS A NOHOLD CONT TO BATCH
&PUNCH /*
&PUNCH /JOB USERAA AA04 BUUPDATE
      * KEEP TRACK OF BATCH EXECUTION OF THIS JOB FOR USER
      * MAKE SURE SYSTEM LIBRARY DISK IS AVAILABLE FOR BATCH
      *   THE SYSTEM DISK HAS THE DISTRIBUTED ISPF LIBRARIES
&PUNCH CP LINK MAINT 19E 19E RR ALL
&PUNCH ACC 19E Y
      * HAVE BATCH LINK TO USER DISKS AS REQUIRED
&PUNCH CP LINK USERAA 191 291 RR USERAAR
&PUNCH ACC 291 B
      * SET UP FILEDEFS TO ACCESS LIBRARIES
&PUNCH FILEDEF ISPLIB DISK ISPLIB MACLIB * (PERM
&PUNCH FILEDEF ISPLIB DISK ISPLIB MACLIB * (PERM
&PUNCH FILEDEF ISPSLIB DISK SKELS MACLIB * (PERM CONCAT
&PUNCH FILEDEF ISPSLIB DISK ISPSLIB MACLIB * (PERM CONCAT
&PUNCH FILEDEF ISPTLIB DISK TABLES MACLIB * (PERM CONCAT
&PUNCH FILEDEF ISPTLIB DISK PROFLIB MACLIB * (PERM CONCAT
&PUNCH FILEDEF ISPTLIB DISK ISPTLIB MACLIB * (PERM CONCAT
&PUNCH FILEDEF ISPPROF DISK PROFILE MACLIB A (PERM
      * INVOKE THE DIALOG MANAGER PASSING THE NAME OF THE COMMAND
&PUNCH EXEC ISPSTART CMD( BUUPDATE )
      * CLEANUP
&PUNCH CP SP PUN TO USERAA
&PUNCH CP SP CONS CLOSE STOP TO USERAA
      * SEND BACK LIST AND LOG FILES
&PUNCH DISK DUMP SPFLOG LISTING A
&PUNCH DISK DUMP SPFLIST LISTING A
      *
&PUNCH CP CLOSE E
&PUNCH CP SPOOL CONSOLE STOP CLOSE
&PUNCH /*
CP SPOOL PUNCH NOCONT CLOSE
*
```

Figure 21. VM/SP Batch Job

---



## VSE Batch Environment

User program dialogs may be executed in a VSE batch environment. The sequential data sets required by ISPF are defined with DLBL/EXTENT/ASSGN JCL. The panel, message, skeleton, and table libraries are defined by ISPDEF control statements. The ISPSTART command specifies the dialog to be executed.

The log data set, although not required for batch execution, should be specified. If the log data file is used, it is kept. It is available to be viewed by use of the ISPF/PDF BROWSE service, if ISPF/PDF is installed.

### Sample Batch Job

Figure 22 shows a sample VSE/AF 1.3.5 batch job. Figure 23 on page 101 shows a sample VSE/AF 2.1 batch job. The ISPSTART command is specified in the VSE background input stream (SYSIN) with the parameter PGM(DIALOG). This program contains the ISPF service requests to be performed.

---

```
* $$ JOB JNM=BATCH,CLASS=0,DISP=D
// JOB BATCH
* DEFINE THE LOG AND TEMPORARY CONTROL DATA SETS
// DLBL ISPLOG,'BATCH.LOG'
// EXTENT SYS001,SERNUM,1,1,3000,10
// DLBL ISPCTL1,'BATCH.TEMP.CONTROL'
// EXTENT SYS001,SERNUM,1,1,4000,10
// ASSGN SYS001,160
// LIBDEF CL,SEARCH=ISPFDMC
// EXEC ISPSTART, SIZE=20K
* SPECIFY DIALOG TO BE EXECUTED
ISPSTART PGM(DIALOG)
* DEFINE VSE LIBRARIES
ISPDEF *,SEARCH=(XYZLIB,ISPFDMC)
/*
/&
* $$ EOJ
```

Figure 22. VSE/AF 1.3.5 Batch Job

---

---

```

* $$ JOB JNM=BATCH,CLASS=0,DISP=D
// JOB BATCH
* DEFINE THE LOG AND TEMPORARY CONTROL DATA SETS
// DLBL ISPLOG,'BATCH.LOG'
// EXTENT SYS001,SERNUM,1,1,3000,10
// DLBL ISPCTL1,'BATCH.TEMP.CONTROL'
// EXTENT SYS001,SERNUM,1,1,4000,10
// ASSGN SYS001,160
// LIBDEF PHASE,SEARCH=ISPF.DM
// EXEC ISPSTART, SIZE=20K
* SPECIFY DIALOG TO BE EXECUTED
ISPSTART PGM(DIALOG)
* SET UP ISPDEF TO ACCESS VSE LIBRARIES/SUBLIBRARIES
ISPDEF *,SEARCH=(XYZLIB.XYZSUB,ISPF.DM)
/*
/&
* $$ EOJ

```

Figure 23. VSE/AF 2.1 Batch Job

---

### Error Processing

ISPF terminates with an error message if a required library is not available. In batch mode, the ISPSTART command must specify invocation of a program function - PGM(program-name) - and if one is not specified, ISPF is not invoked and a message, issued on SYSLOG, states that display services are not available in a batch environment.

Any errors, including severe errors, that occur during batch mode execution are handled in the same way as errors encountered during ICCF execution. In both instances, severe error messages are written to SYSLOG.

If a function abends, an IDUMP is taken and the ISPF batch job is terminated.



## CHAPTER 6. DESCRIPTION OF SERVICES

This chapter contains a description of syntax conventions and return codes for the dialog services, followed by a detailed description of each service. The service descriptions are arranged in alphabetic order for ease of reference. Appendix E, "Summary of ISPF Syntax," contains a quick reference summary of dialog services. (See ISPF Dialog Management Services Examples for examples of invoking ISPF services in application dialogs.) Notation conventions are described in the Preface.

### INVOCATION OF SERVICES

Each service description shows the format for command invocation and for call invocation from a program module. The command invocation format is used from a CLIST, EXEC or, option 7.6 of PDF (the dialog test facility). In VSE, the command invocation format may be used only while in option 7.6 of PDF.

Call invocation formats are shown in PL/I syntax. For example, ";" ends statements in the formats described. This is a PL/I convention, but syntax should be appropriate to the language being used.

Included in each service description is an example of its use in the command procedure format and the PL/I call format. Additional examples, including COBOL and FORTRAN call formats, may be found in ISPF Dialog Management Services Examples.

If ISPF/PDF is installed, consider using its model facilities (under edit) when coding requests for ISPF services (see ISPF/PDF Reference for a description of these facilities).

### Command Invocation

ISPF services are invoked using the ISPEXEC command in a command procedure (CLIST or EXEC) or while operating under option 7.6 of ISPF/PDF.

The general format for command invocation is:

---

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

---

The "service-name" is alphabetic, up to eight characters long.

For some services, "parameter1" is a positional parameter and is required. 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.

In MVS and VM/SP, CLIST or EXEC 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.

#### Notes:

1. In MVS, TSO CLIST attention exits are not recognized by ISPF and, if used, may cause unpredictable results.
2. In VM/SP, EXEC variables appearing within parentheses must be followed by a blank, preceding the closing parenthesis. For example:

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

Some ISPF services allow the names of dialog variables to be passed as parameters. (Variable names are eight characters or less). These names should not be preceded with an ampersand unless substitution is desired; for example:

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

*- will substitute*

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 [ASIS|SHARED|PROFILE]
```

In this case, "name-list" is a positional parameter. It may consist of a list of one or more (up to 254) dialog variable names, each name separated by commas or blanks. If the name-list consists of more than one name, it must be enclosed in parentheses. Parentheses may be omitted if a single name constitutes the list; for example:

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

The last two lines of the example (with and 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 TBPOT service is:

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

### VM/SP: Using the &PRESUME Statement

In VM/SP, the following statement may be included in an EXEC 2 procedure prior to issuing the first ISPEXEC command:

```
&PRESUME &SUBCOMMAND ISPEXEC
```

This statement allows the omission of "&SUBCOMMAND" and "ISPEXEC" in requests for ISPF services.

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

If &PRESUME &SUBCOMMAND ISPEXEC statement is not included, every request for ISPF services must be preceded by &SUBCOMMAND ISPEXEC, as follows:

```
&TRACE OFF
&SUBCOMMAND ISPEXEC DISPLAY PANEL(ABC)
```

&TRACE specifies the use of the EXEC 2 language.

This last mentioned form is used in this publication, but the '&SUBCOMMAND' is not shown with the service.

### Call Invocation

ISPF services are invoked from programs, except FORTRAN programs, by calling a subroutine named ISPLINK. FORTRAN programs must invoke this subroutine using another name, ISPLNK, because, in FORTRAN, the maximum length of a module name is six characters.

**Note:** Only a single task level is permitted. In MVS, a dialog function may attach a lower-level subtask, but the subtask may not invoke ISPF services. User dialogs may not use storage subpools 33 through 48, because these pools are reserved for use by ISPF.

Examples of service requests in programs (COBOL, FORTRAN, and PL/I) may be found in ISPF Dialog Management Services Examples.

The general format for invoking ISPF services from functions, other than FORTRAN, is:

---

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

---

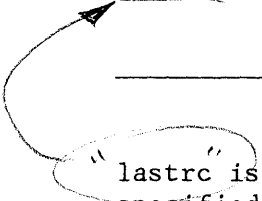
The parameters in call statements are all positional; they must be coded in the order described for each service. Optional parameters may be omitted in a right-to-left dropout sequence. To obtain the default value for an optional parameter, code the parameter as one or more blanks. This has the same effect as omitting the parameter and is used when parameters are to be dropped in other than a right-to-left sequence.

The general format for invoking ISPF services from FORTRAN functions is:

---

```
lastrc = ISPLNK (service-name, parameter1, parameter2 ...)
```

---



"lastrc" is a FORTRAN integer variable in which the return code from the specified ISPF service is available. "lastrc" is any valid FORTRAN name.

For functions written in FORTRAN, arguments may be passed as FORTRAN variables or literals.

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, COBOL, and FORTRAN call statements. It requires use of the VL keyword in Assembler call statements.

Call statements are shown in PL/I syntax. Service names and keyword values are shown as literals, enclosed in apostrophes ('); for 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.

In PL/I programs, the following declare statements should be included:

```
DECLARE ISPLINK /*NAME OF ENTRY POINT*/  
        ENTRY  
        EXTERNAL /*EXTERNAL ROUTINE*/  
        OPTIONS( /*NEEDED OPTIONS*/  
        ASM, /*DO NOT USE PL/I DOPE VECTORS*/  
        INTER, /*INTERRUPTS*/  
  
        RETCODE); /*EXPECT A RETURN CODE*/
```

**Note:** In VSE, RETCODE should not be specified.

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, as in the following examples:

**PL/I example:**

```
DECLARE SERVICE CHAR(8) INIT('TBOPEN '),  
        TABLE CHAR(8) INIT('XTABLE '),  
        OPTION CHAR(8) INIT('NOWRITE ');  
        . . .  
CALL ISPLINK (SERVICE, TABLE, OPTION);
```

**COBOL example:**

```
WORKING-STORAGE SECTION.  
    77 SERVIS      PICTURE A(8) VALUE 'TBOPEN ' .  
    77 TABL        PICTURE A(8) VALUE 'XTABLE ' .  
    77 OPTSHUN     PICTURE A(8) VALUE 'NOWRITE ' .  
    . . .  
  
PROCEDURE DIVISION.  
    CALL 'ISPLINK' USING SERVIS TABL OPTSHUN.
```

**FORTRAN example:**

```
INTEGER SERVICE(2),TABLE(2),OPTION(2)  
DATA SERVICE/'TBOP','EN '/  
DATA TABLE/'XTAB','LE '/  
DATA OPTION/'NOWR','ITE '/  
    . . .  
  
LASTCC=ISPLNK(SERVICE, TABLE, OPTION)
```



For service calls in PL/I and COBOL parameter variables may be initialized using literals in assignment statements, as in the following examples:

**PL/I example:**

```
SERVICE='TBOPEN';
```

**COBOL example:**

```
MOVE "TBOPEN" TO SERVICE.
```

**FORTRAN example:** For FORTRAN service requests, previously defined constants must be used in assignment statements; for example:

```
INTEGER TBOPEN(2),SERVICE(2)  
DATA TBOPEN/'TBOP','EN '/
```

```
SERVICE=TBOPEN
```

## Parameters

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

- 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 eight 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 eight characters. The exceptions are data set name, volume serial, and fileid (see the description of the EDIT and BROWSE services).
- Numeric value: A fullword signed binary number.
- Name list - string format:. A list of dialog variable names coded as a character string. Each name is one to eight characters. 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; for example:

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

When the list consists of a single name, the parentheses are not required, but a trailing blank is required if the name is less than eight characters in length.

- Name list - structure format:. A list of dialog variable names passed in a structure. Each name is one to eight characters. 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 a value of either zero or eight.
  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

Return codes from services 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 operation, due to errors.

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

- CANCEL - displays and logs a message, then terminates the dialog and redisplay the primary option menu.
- RETURN - formats an error message (but does not display or log it), then returns to the function that invoked the service, passing back the designated return code.

The dialog may set the error mode by means of the CONTROL service. The default mode is CANCEL. In CANCEL mode, generally, control is not returned to the function that invoked the service. Hence, the function generally will not see a return code of 12 or higher, and may not include logic to process these kinds of errors. However, this is not so for a return code of 20 from ISPLINK, when it is caused by an invalid ISPF environment. In this case, because ISPF is not capable of displaying an error panel (or any panel), control is returned to the dialog, even though the return code is 20.

In RETURN mode, control is 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 with 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. ZERRMSG, ZERRSM, and ZERRLM are changed only when the return code from an ISPF service is greater than 8. If a corresponding help panel was specified in the message definition, the name of the help panel is contained in system variable ZERRHM. All of these system variables are in the shared variable pool.

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 )
```

The short and long message text and the name of the corresponding help panel are provided in the event that other action is desired.

## MVS and VM/SP: 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 EXEC 2 variable RETCODE. For call invocation, the code is returned in register 15 or, in FORTRAN programs, in registers 15 and 0.

Programs coded in FORTRAN may examine the return code by using an integer variable, such as lastrc in the following example:

```
lastrc = ISPLNK (service name, parameter1, parameter2 ...)
```

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 INTER RETCODE);
DECLARE PLIRETV BUILTIN;
```

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

## VSE: Return Codes and Other Processing Considerations

This section describes the use of return codes and other processing considerations when preparing a dialog, written in a high-level language, to run in VSE.

### FORTRAN

Programs coded in FORTRAN may examine the return code by using an integer variable, such as `lastrc` in the following example:

```
lastrc = ISPLNK (service name, parameter1, parameter2 ...)
```

Programs written at language level 77, must specify the compiler option `SC` for `ISPLNK`. This creates a parameter list acceptable to `ISPF`.

`ISPF` does not permit the use of FORTRAN programs that use phase overlay structures.

### PL/I

When invoking a PL/I program by use of the `ISPSTART` command or the `SELECT` service, the `LANG` keyword must be specified with either the `PLI` or `PL1` parameter, e.g., `LANG(PLI)`. A second parameter, storage-area, may also be specified to indicate the number of storage bytes to be made available to the PL/I program being invoked. (The default storage size is 2816 bytes. Generally, the amount of storage required for a PL/I program can be determined by compiling the program using the storage option and adding 2192 to the DSA sizes indicated on the compile listing. However, the storage requirement will vary considerably depending on the compiler options specified; for example: while "FLOW" requires a small amount of additional storage, "COUNT" adds substantially to required storage).

PL/I programs may examine the return code from `ISPF` services by using the `PLIRETV` built-in function.

The following declare statements are required for PL/I programs:

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

PL/I programs must be link edited with modules `ISPPLI` and `IBMBPJRA`. `ISPPLI` must be the first physical module in the phase and `IBMBPJRA` must be the second physical module. In addition, linkage editor control statements must include a statement defining the entry point of the program; for example:

```
PHASE EMPLFN,*  
INCLUDE ISPLI  
INCLUDE IBMBPJRA  
INCLUDE EMPLFN  
INCLUDE ISPLINK  
ENTRY EMPLFN
```

ISPF does not permit the use of PL/I programs that use phase overlay structures.

## COBOL

When invoking a COBOL program by use of either the ISPSTART command or the SELECT service, the LANG keyword must be specified with the COBOL parameter, e.g., LANG(COBOL).

To examine the return code from ISPF services, COBOL programs must have a LINKAGE section with the return code field coded as the first item in the section; for example:

```
LINKAGE SECTION  
77 RETURN-CODE PIC 9(8) COMP-4.
```

Although the name of the return code field can be any name valid in COBOL, it is recommended that RETURN-CODE be used to make the dialog compatible with MVS and VM/SP. Regardless of what name is used, the data description must appear exactly as shown.

ISPF does not permit the use of COBOL programs that use phase overlay structures.

## SERVICES

This section describes the ISPF services. The services are presented in alphabetic order. For each service, the command procedure format is shown, followed by the PL/I call format. Following this, the parameters used when invoking the service are described.

## BROWSE - MVS or VM/SP: Display a Data Set or File

*not the CMS (ISPF) Browse feature*

The BROWSE service provides an interface to the ISPF/PDF browse program, bypassing display of the browse entry panel. (Use of BROWSE requires installation of ISPF/PDF.) The BROWSE service may not be issued by a PL/I main program that also uses subtasking. See ISPF/PDF 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/SP 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 is 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 is displayed.

The maximum length of the dsname parameter is 56 characters.

**serial**

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

The maximum length of this 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 is displayed.

Nested dialogs may not use the BROWSE service if any of the functions listed below are active under the same logical screen. Violation of this restriction results in a severe error.

- ISPF/PDF option 1 (BROWSE)
- ISPF/PDF option 3 (UTILITIES)
- ISPF/PDF option 4 (FOREGROUND)
- User functions that invoke the BROWSE service

The following return codes are possible:

- 0 - Normal completion
- 20 - Severe error

Example:

1. MVS

Invoke the BROWSE service to allow browsing of TELOUT, a member of the ISPFPROJ.FTOUTPUT data set.

```
ISPEXEC BROWSE DATASET('ISPFPROJ.FTOUTPUT(TELOUT)')
```

```
CALL ISPLINK('BROWSE', 'ISPFPROJ.FTOUTPUT(TELOUT)');
```

2. VM/SP

Invoke the BROWSE service to allow browsing of TELOUT, a member of the FTOUTPUT MACLIB maclib.

```
ISPEXEC BROWSE FILE(FTOUTPUT MACLIB *) MEMBER(TELOUT)
```

```
CALL ISPLINK('BROWSE', '(FTOUTPUT MACLIB *)', 'TELOUT');
```



## BROWSE - VSE: Display a Library or File

The BROWSE service provides an interface to the ISPF/PDF browse program, bypassing display of the browse entry panel. (Use of BROWSE requires installation of ISPF/PDF.) The BROWSE service may not be issued by a PL/I main program that also uses subtasking. See ISPF/PDF Reference for a description of BROWSE.

---

```
BROWSE DATASET(dsname) [RECFORM(recfm)]
                        [RECSIZE(lrecl)]
                        [BLKSIZE(blksize)]
                        [DEVADDR(sysno)]
                        [VOLUME(serial)]
```

```
CALL ISPLINK ('BROWSE', dsname [,recfm]
              [,lrecl]
              [,blksize]
              [,sysno]
              [,serial] );
```

---

| **dsname** (for VSE/AF 1.3.5)  
| Specifies one of the following:

- 1111111.s.mmmmmmmm.tt
- 1111111..mmmmmmmm.tt
- 1111111.s..tt
- 1111111...tt
- nnnn
- nnnn.iiiiiii
- nnnn.iiiiiii.pppp
- fffffff

where:

- l's represent an AF library name; maximum length is 7 characters
- s represents the AF sublibrary (source libraries only); length is 1 character
- m's represent the AF member name; maximum length is 8 characters
- tt represents the AF library type (SL, RL, PL),
- n's represent a VSE/ICCF library number; maximum length is 4 digits
- i's represent the VSE/ICCF library member name; maximum length is 8 characters
- p's represent the VSE/ICCF library member password; maximum length is 4 characters
- f's represent a VSE sequential fileid; maximum length is 7 characters

**Note:** If a member name is not specified, a selection list is provided.

**dsname (for VSE/AF 2.1)**

Specifies one of the following:

- llllllll.ssssssss.mmmmmmmm.tttttttt
- llllllll.ssssssss.mmmmmmmm
- llllllll.ssssssss..tttttttt
- nnnn
- nnnn.iiiiiiii
- nnnn.iiiiiiii.pppp
- fffffff

where:

- l's represent a VSE/AF library name; maximum length is 7 characters
- s's represents a VSE/AF sublibrary; maximum length is 8 characters

- m's represent the VSE/AF member name; maximum length is 8 characters
- t's represent the VSE/AF member type; maximum length is 8 characters
- n's represent a VSE/ICCF library number; maximum length is 4 digits
- i's represent the VSE/ICCF library member name; maximum length is 8 characters
- p's represent the VSE/ICCF library member password; maximum length is 4 characters
- f's represent a VSE sequential fileid; maximum length is 7 characters

**Note:** If a member name is not specified, a selection list is provided.

#### **recfm**

Specifies the record format of the VSE sequential data set, as follows:

- F - specifies fixed format
- V - specifies variable format
- U - specifies undefined format

#### **lrecl**

Specifies, in five digits or less, the record size of the sequential data set. If called from a program, lrecl should be declared as a fullword variable and set to the correct value.

#### **blksize**

Specifies, in five digits or less, the block size of the sequential data set. If called from a program, blksize should be declared as a fullword variable and set to the correct value.

#### **sysno**

Specifies the three numerical digits of the programmer logical unit on which the sequential data set is mounted.

#### **serial**

Specifies, in six characters or less, the serial number of the volume on which the data set resides.

Nested dialogs may not use the BROWSE service if any of the functions listed below are active under the same logical screen. Violation of this restriction results in a severe error.

- ISPF/PDF option 1 (BROWSE)

- ISPF/PDF option 3 (UTILITIES)
- User functions that invoke the BROWSE service

The following return codes are possible:

- 0 - Normal completion
- 20 - Severe error

Example:

1. Under VSE/AF 1.3.5, invoke the BROWSE service to allow browsing of TELOUT, a member of source statement library USERSRC.

```
CALL ISPLINK ('BROWSE', 'USERSRC.S.TELOUT.SL ');
```

2. Under VSE/AF 2.1, invoke the BROWSE service to allow browsing of TELOUT, a skeleton member of library.sublibrary USERLIB.USER1.

```
CALL ISPLINK ('BROWSE', 'USERLIB.USER1.TELOUT.S');
```

3. Invoke the BROWSE service to allow browsing of a sequential data set named SEQFILE, consisting of fixed length 80-byte records, blocked 10, mounted on SYS001, volume number 11111. Program variables RSIZE and BSIZE are full words and contain the binary equivalent of 80 and 800, respectively.

```
CALL ISPLINK ('BROWSE', 'SEQFILE ', 'F', RSIZE, BSIZE,
              '001', '11111');
```

## CONTROL - Set Processing Modes

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

---

```
ISPEXEC CONTROL { DISPLAY { LOCK
                        LINE [START(line-number)]
                        SM  [START(line-number)]
                        REFRESH
                        SAVE|RESTORE
                    }
                }
                { 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 'LOCK', 'LINE', 'SM', 'REFRESH',

'SAVE', or 'RESTORE' 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 LOCK, LINE, SM, REFRESH, SAVE and RESTORE. LINE and SM are in effect until the next display of an ISPF panel. REFRESH occurs on the next display of an ISPF panel.

### LOCK

Specifies that the next (and only the next) display output (e.g., by the DISPLAY or TBDISPL service) is to leave the terminal user's

keyboard locked. ISPF processes the next display output as though the user had depressed the ENTER key.

This facility may be used to display an "in process" message during a long running operation. It is the dialog developer's responsibility to ensure that the keyboard is unlocked (by the subsequent display of a message or panel).

## LINE

Specifies that terminal line-mode output is expected (e.g., from a TSO or CMS command or VSE system or program dialog). The screen is completely rewritten on the next ISPF full screen write operation, after the line(s) have been written.

**Note:** CONTROL DISPLAY LINE is automatically invoked by the SELECT service whenever a SELECT CMD request is encountered, unless the command begins with a percent (%) sign; for example:

SELECT CMD(ABC) - causes automatic entry into line mode.  
SELECT CMD(%ABC) - no automatic entry into line mode.

## line-number

In MVS, 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. The parameter should specify a line value that is not within 3 lines of the bottom of the logical screen. If the value is within 3 lines of the bottom of the logical screen, a default line value is used. This value is equivalent to the number of the bottom line of the screen, minus 3.

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. Once line mode has been set, subsequent attempts to set line mode (without intervening full screen output) are ignored. Accordingly, the line-number, once set, cannot be changed.

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

In VSE, this parameter is ignored. Line mode output is always displayed immediately following the ICCF column indicator line.

## 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 ISPF-generated full screen write is issued to the terminal. This facility should be used before or after invoking any program that uses non-ISPF services for generating full screen output (e.g., XEDIT).

## SAVE

Used in conjunction with DISPLAY, TBDISPL, BROWSE, or EDIT processing, specifies that information about the current logical screen (including control information) is to be saved.

Use of the CONTROL service SAVE and RESTORE allows DISPLAY, TBDISPL, BROWSE, or EDIT processing to be nested. The CONTROL service is used to save and restore the environment at each level. Nesting of successive requests for the same service is not allowed.

**Note:** Whenever the dialog manager encounters a SELECT command entered by the user in the command field of a displayed panel as described in "Command Table Format" on page 54, the current display environment is automatically saved prior to invoking the designated dialog. That environment is subsequently restored when the dialog ends.

Certain positioning information, including the ZTDTOP variable and the current row pointer, is not saved. The variable ZVERB is not saved.

## RESTORE

Specifies the restoration of information previously saved by CONTROL DISPLAY SAVE. The logical screen image is restored exactly as it appeared when the SAVE was performed. Processing of the previous panel or table display can then be resumed.

## 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.

## 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 command 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 using this, the CONTROL, service.

## CANCEL

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

## RETURN

Specifies that control should be returned to the dialog on an error. The system variable ZERRMSG contains the message id for a message that describes the error. The message is not written to the ISPF log file (unless TRACE mode is in effect), nor is an error panel displayed.

*→ page 89*  
**Note:** If a dialog developer wants, on an error, to ABEND with STAE, he must specify CONTROL RETURN because specification of CONTROL CANCEL can nullify the developer's requested STAE.

## SPLIT

Defines the user's ability to enter split screen mode.

## ENABLE

Specifies that the user is to be allowed to enter split screen mode. Split screen mode is normally enabled. It is disabled only if explicitly requested by use of the CONTROL service. It remains disabled until explicitly re-enabled by the CONTROL service.

## 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.

The following return codes are possible:

- 0 - Normal completion.
- 8 - Split screen mode already in effect (applies only to a SPLIT DISABLE request); split screen remains enabled.
- 20 - Severe error.

Example: Set the error processing mode to allow the dialog function to process return codes of 12 or higher.

```
ISPEXEC CONTROL ERRORS RETURN
```

```
CALL ISPLINK('CONTROL', 'ERRORS ', 'RETURN ');
```



## 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.

The user may enter information in fields specified to be input fields on the panel definition. After the user presses ENTER, content of the input fields is stored in dialog variables specified on the panel definition. Then, any processing specified on the panel definition is performed and the DISPLAY service returns to the calling function.

---

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

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

---

### **panel-name**

Specifies the name of the panel to be displayed.

### **message-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.

Regardless of the format, all of the parameters are optional. The processing of the panel-name and message-id parameters is as follows:

- If panel-name is specified and message-id is not specified, the panel is read from the panel library, initialized, and displayed without a message.
- If panel-name and message-id are both specified, the panel is read from the panel library, initialized, and displayed with the specified message.

- If panel-name is not specified and message-id is specified, the current panel is overlaid with a message, without any initialization being performed on the panel.
- If neither panel-name nor message-id is specified, the current panel is 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 "Panel Processing Considerations" in Chapter 7.

The following return codes are possible:

- 0 - Normal completion.
- 8 - User requested termination via the END or RETURN command.
- 12 - The specified panel, message, or cursor field could not be found.
- 16 - Truncation or translation error in storing defined variables.
- 20 - Severe error.

Example: (See Appendix A, "Using the DISPLAY Service" on page 261 for another example of DISPLAY processing).

Panel definition XYZ specifies display of variables AAA and KLM (an input field). Using this definition, invoke ISPF services to display these variables at the terminal and superimpose, on line 1, the short form text of message number ABCX013. Position the cursor, on the display, at the beginning of input field KLM ready for entry of data by the person at the terminal.

```
ISPEXEC DISPLAY PANEL(XYZ) MSG(ABCX013) CURSOR(KLM)
```

```
CALL ISPLINK('DISPLAY', 'XYZ ', 'ABCX013 ', 'KLM ');
```

## EDIT - MVS or VM/SP: EDIT a Data Set or File

The EDIT service provides an interface to the ISPF/PDF editor, bypassing display of the edit entry panel. (Use of EDIT requires installation of ISPF/PDF.) The EDIT service may not be issued by a PL/I main program that also uses subtasking. See ISPF/PDF 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/SP 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 is 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 is 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 is searched for the data set name.

The maximum length of this 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/SP environment, if fileid specifies a non-existent file, the user is able to create a new file. However, the file characteristics (record format and logical record length) may be unpredictable. They are 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 are 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 is displayed.

Nested dialogs may not use the EDIT service if any of the functions listed below are active under the same logical screen. Violation of this restriction results in a severe error.

- ISPF/PDF option 2 (EDIT)
- ISPF/PDF option 3 (UTILITIES)
- ISPF/PDF option 4 (FOREGROUND)
- User functions that invoke the EDIT service

*Recursive  
error* →

The following return codes are possible:

0 - Normal completion, data was saved.

4 - Normal completion, data was not saved.

**Note:** For a D37 space ABEND, a partial save is done, up to the point where space was depleted.

20 - Severe error.

Example:

1. MVS

Invoke the EDIT service to allow editing of TELOUT, a member of the ISPFPROJ.FTOUTPUT data set.

```
ISPEXEC EDIT DATASET(ISPFPROJ.FTOUTPUT(TELOUT))
```

```
CALL ISPLINK('EDIT', 'ISPFPROJ.FTOUTPUT(TELOUT)');
```

2. VM/SP

Invoke the EDIT service to allow editing of TELOUT, a member of the FTOUTPUT MACLIB maclib.

```
ISPEXEC EDIT FILE(FTOUTPUT MACLIB *) MEMBER(TELOUT)
```

```
CALL ISPLINK('EDIT', '(FTOUTPUT MACLIB *)', 'TELOUT');
```

## EDIT - VSE: EDIT a Library or File

The EDIT service provides an interface to the ISPF/PDF editor, bypassing display of the edit entry panel. (Use of EDIT requires installation of ISPF/PDF.) The EDIT service may not be issued by a PL/I main program that also uses subtasking. See ISPF/PDF Reference for a description of the editor.

---

```
EDIT DATASET(dsname) [PROFILE(profile)]
                        [RECFORM(recfm)]
                        [RECSIZE(lrecl)]
                        [BLKSIZE(blksize)]
                        [DEVADDR(sysno)]
                        [VOLUME(serial)]
```

```
CALL ISPLINK ('EDIT', dsname [,profile]
              [,recfm]
              [,lrecl]
              [,blksize]
              [,sysno]
              [,serial] );
```

---

**dsname** (for VSE/AF 1.3.5)  
Specifies one of the following:

- 1111111.s.mmmmmmmm.tt
- 1111111..mmmmmmmm.tt
- 1111111.s..tt
- 1111111...tt
- nnnn
- nnnn.iiiiiii

- nnnn.iiiiiii.pppp
- fffffff

where:

- l's represent an AF library name; maximum length is 7 characters
- s represents the AF sublibrary (source libraries only); length is 1 character
- m's represent the AF member name; maximum length is 8 characters
- tt represents the AF library type (SL, RL, PL),
- n's represent a VSE/ICCF library number; maximum length is 4 digits
- i's represent the VSE/ICCF library member name; maximum length is 8 characters
- p's represent the VSE/ICCF library member password; maximum length is 4 characters
- f's represent a VSE sequential fileid; maximum length is 7 characters

**Note:** If a member name is not specified, a selection list is provided.

#### dsname (for VSE/AF 2.1)

Specifies one of the following:

- lllllll.sssssss.mmmmmmm.ttttttt
- lllllll.sssssss.mmmmmmm
- lllllll.sssssss..ttttttt
- nnnn
- nnnn.iiiiiii
- nnnn.iiiiiii.pppp
- fffffff

where:

- l's represent a VSE/AF library name; maximum length is 7 characters

- s's represents a VSE/AF sublibrary; maximum length is 8 characters
- m's represent the VSE/AF member name; maximum length is 8 characters
- t's represent the VSE/AF member type; maximum length is 8 characters
- n's represent a VSE/ICCF library number; maximum length is 4 digits
- i's represent the VSE/ICCF library member name; maximum length is 8 characters
- p's represent the VSE/ICCF library member password; maximum length is 4 characters
- f's represent a VSE sequential fileid; maximum length is 7 characters

**Note:** If a member name is not specified, a selection list is provided.

**profile**

Specifies the name of the edit profile. When omitted, the default is ISPEDIT.

**recfm**

Specifies the record format of the VSE sequential data set, as follows:

F - specifies fixed format

V - specifies variable format

**lrecl**

Specifies, in five digits or less, the record size of the sequential data set. If called from a program, lrecl should be declared as a fullword variable and set to the correct value.

**blksize**

Specifies, in five digits or less, the block size of the sequential data set. If called from a program, blksize should be declared as a fullword variable and set to the correct value.

**sysno**

Specifies the three numerical digits of the programmer logical unit on which the sequential data set is mounted.

**serial**

Specifies the volume serial, six or less characters, on which the sequential data set resides.



Nested dialogs may not use the EDIT service if any of the functions listed below are active under the same logical screen. Violation of this restriction results in a severe error.

- ISPF/PDF option 2 (EDIT)
- ISPF/PDF option 3 (UTILITIES)
- User functions that invoke the EDIT service

The following return codes are possible:

0 - Normal completion.

20 - Severe error.

Example:

1. Under VSE/AF 1.3.5, invoke the EDIT service to allow browsing of TELOUT, a member of source statement library USERSRC.

```
CALL ISPLINK ('EDIT', 'USERSRC.S.TELOUT.SL')
```

2. Under VSE/AF 2.1, invoke the EDIT service to allow browsing of TELOUT, a skeleton member of library.sublibrary USERLIB.USER1.

```
CALL ISPLINK ('EDIT', 'USERLIB.USER1.TELOUT.S')
```

3. Invoke the EDIT service to allow editing of a sequential data set named SEQFILE, consisting of fixed length 80-byte records, blocked 10, mounted on SYS001, volume number 11111. Program variables RSIZE and BSIZE are full words and contain the binary equivalent of 80 and 800, respectively. Use the default profile.

```
CALL ISPLINK ('EDIT', 'SEQFILE ', ' ', 'F', RSIZE, BSIZE,  
              '001', '11111');
```

*batch-5/16/64/10-5*

## FTCLOSE - End File Tailoring

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

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

The library parameter should be specified if a library other than that represented by the ISPFILDEF definition is to be used. The library parameter is ignored if the "TEMP" option is specified on the FTOPEN service or if the ISPFILDEF definition specifies a sequential data set. A severe error occurs if file tailoring attempts to use the data set and it is not a library.

The NOREPL parameter specifies that an existing member in the file tailoring output library is not to be overlaid (replaced) by the current FTCLOSE service. If a member of the same name already exists, the FTCLOSE service request is terminated with a return code of 4 and the original member remains unaltered.

---

```
ISPEXEC FTCLOSE [NAME(member-name) [LIBRARY(library-name)]  
                [NOREPL]
```

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

---

### **member-name**

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

### **library-name**

Specifies the name of a DD, FILEDEF, or ISPDEF statement that defines the output library in which the member-name exists. ISPFILDEF is the default if this parameter is omitted.

### **NOREPL**

Specifies that FTCLOSE is not to overlay an existing member in the output library.

The following return codes are possible:

- 0 - Normal completion.
- 4 - Member already exists in the output library and NOREPL was specified. The original member is unchanged.
- 8 - File not open (FTOPEN was not used prior to FTCLOSE).
- 12 - Output file in use; ENQ failed.
- 16 - Skeleton library or output file not allocated.
- 20 - Severe error.

Example: End the file tailoring process and store the result of the processing in the file tailoring output library in member TELOUT.

```
ISPEXEC FTCLOSE NAME(TELOUT)
```

```
CALL ISPLINK('FTCLOSE', 'TELOUT');
```

## FTERASE - Erase File Tailoring Output

The FTERASE service erases (deletes) a member of a file tailoring output library.

A severe error occurs if a specified library or the default (ISPFIL) is a sequential file.

---

```
ISPEXEC FTERASE member-name [LIBRARY(library-name)]
```

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

---

### **member-name**

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

### **library-name**

Specifies the name of a DD, FILEDEF, or ISPDEF statement that defines the output library in which the member-name to be deleted exists. ISPFIL is the default if this parameter is omitted.

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.

Example: Erase member TELOUT in the file tailoring output library.

```
ISPEXEC FTERASE TELOUT
```

```
CALL ISPLINK('FTERASE', 'TELOUT');
```

*batch skeletons*

## FTINCL - Include a 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.

See "Skeleton Definitions" in Chapter 7.

---

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

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

---

### skel-name

Specifies the name of the skeleton.

### NOFT

Specifies that no file tailoring is to be performed on the skeleton: the entire skeleton is to be copied to the output file exactly as is with no variable substitution or interpretation of control records.

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 or output file not allocated.
- 20 - Severe error.

Example: Perform file tailoring using the file tailoring skeleton named TELSKEL, a member in the file tailoring skeleton library, to control processing.

```
ISPEXEC FTINCL TELSKEl
```

```
CALL ISPLINK('FTINCL', 'TELSKEL ');
```

*batch skeleton*

## FTOPEN - Begin File Tailoring

The FTOOPEN 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 ISPF. ISPSLIB may specify a concatenation of libraries. See Chapter 4, "Library Requirements," for library setup requirements.

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

---

```
ISPEXEC FTOOPEN [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 ISPF. Its name is available in system variable ZTEMPF.

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

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

```
SUBMIT '&ZTEMPF'
```

**Note:** If ISPCTL1 and ISPCTL2 are preallocated to VIO, this temporary data set may not be accessed using BROWSE or EDIT.

In VM/SP, the temporary file is written to the user's A-disk. The ISPF-generated file name is contained in ZTEMPF. The file type is always ISPTMP. Data in this file may be sent to another virtual machine using the following CMS command:

```
PUNCH &ZTEMPF ISPTMP
```

In VSE, ZTEMPF contains the filename of the temporary sequential data set.

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 or output file not allocated.
- 20 - Severe error.

Example: Prepare for access (open) both the file tailoring skeleton and file tailoring output libraries.

```
ISPEXEC FTOPEN
```

```
CALL ISPLINK('FTOPEN');
```

## LOG - Write a Message to the Log File

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

---

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

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

---

### **message-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.
- 12 - The message-id contains invalid syntax or was not found.
- 20 - Severe error.

### Example:

- In a CLIST or EXEC, dialog variable TERMSG contains a message-id. Write this message in the ISPF log file.

```
ISPEXEC LOG MSG(&TERMSG )
```

- In a PL/I program, program variable TERMSG contains a message-id. Write this message in the ISPF log file.

```
CALL ISPLINK('LOG', TERMSG);
```

- Write message ABCX013 in the ISPF log file.

```
ISPEXEC LOG MSG(ABCX013)
```

```
CALL ISPLINK('LOG', 'ABCX013 ');
```



↖ Análogous to using &ZSEL within  
a selection menu. (see p. 229)

## SELECT - Select a Panel or Function

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

Within a dialog function a program may invoke another program using standard CALL or link conventions. These are nested programs and are transparent to the dialog manager. On the other hand, when the invoked program is a new dialog function, SELECT must be used.

**Note:** Programs or command processors that use VMCF to communicate with their own disconnected virtual machine, may not be invoked under ISPF.

---

```
ISPEXEC SELECT { PANEL(panel-name) [OPT(option)]
                { CMD(command)
                { PGM(program-name) [PARM(parameters)]
                }
                }
                [NEWAPPL [(application-id)]|NEWPOOL]
```

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

Note: parameters which may appear in buffer are:

```
{ PANEL(panel-name) [OPT(option)]
  { CMD(command)
  { PGM(program-name) [PARM(parameters)]
    [LANG(PLI|PL1 [,storage-area])]
    [LANG(COBOL)]
  }
  }
  [NEWAPPL [(application-id)]|NEWPOOL]
```

---

### panel-name

Specifies the name of a selection panel to be displayed.

### option

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

(Simulated selection)

## command

In MVS and VM/SP, specifies a command procedure (CLIST or EXEC 2), 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 EXEC 2) to improve performance, and to avoid automatic entry into line display mode (see description of CONTROL service).

(EXECNAME P1 P2 )

**Note:** In MVS, ordinary commands (command processors) are invoked by the ATTACH macro and may not issue requests for ISPF dialog services.

In VM/SP, references made by CMS commands while running ISPF, generally, give unpredictable results.

In VSE, this parameter is not used because ICCF procedures may not be used for writing dialog functions.

## 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.

**Note:** Dialog developers should avoid the ISP and ISR prefixes (the ISPF and PDF component codes) in naming dialog functions. Special linkage conventions, intended only for internal ISPF use, are used to invoke programs named "ISPxxxxx" and "ISRxxxxx".

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

In VM/SP, this parameter may specify the name of a TEXT file, a member of a TXTLIB, or a member of a LOADLIB. For more information, see "Library Setup - VM/SP Environment" in Chapter 4.

In VSE/AF 1.3.5, this parameter must specify the name of a phase contained in a core image library. This library must be defined by a LIBDEF CL statement.

In VSE/AF 2.1, this parameter must specify the name of a phase contained in a library.sublibrary, which must be defined by a LIBDEF PHASE statement.

## 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 in 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 ISPF dialog variable, it must be assigned to a fixed character string, because the VDEFINE service cannot handle varying length PL/I strings.

In MVS and VM/SP, 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.

#### LANG(PLI) or LANG(PL1)

In VSE, specifies that the function being invoked is written in the PL/I language.

In MVS and VM/SP, this keyword is not used.

#### storage-area

In VSE, for programs written in PL/I, specifies the number of bytes of dynamic storage to be made available to the function being invoked.

The default storage size is 2816 bytes. Generally, the amount of storage required for a PL/I program can be determined by using the storage option to compile the program and then adding 2192 to the DSA sizes specified on the compile listing. However, the storage requirement will vary considerably depending on the compiler options specified, i.e. "FLOW" will use slightly more storage but "COUNT" will substantially increase the storage requirement.

In MVS and VM/SP, this parameter is not used.

#### LANG(COBOL)

In VSE, specifies that the function being invoked is written in the COBOL language.

In MVS and VM/SP, this keyword is not used.

#### NEWAPPL

Specifies that a new application is being invoked.

#### application-id

Specifies a 1- to 4-character code for the new application named in this SELECT service request. The code is to be prefixed to the user's profile, the edit profile, or the command table associated with the application, as follows (where xxxx is the application-id):

Application Profile - xxxxPROF  
Edit Profile - xxxxEDIT  
Command Table - xxxxCMDS

These are table (member) names in the profile or table input library.

If the NEWAPPL keyword is specified but the application-id is not specified, the default application-id is ISP, as follows:

User Profile - ISPPROF  
Edit Profile - ISPEDIT  
Command Table - ISPCMDS

### 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 is reinstated.

### buf-length

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

### buffer


Specifies the name of 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; for example:

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

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 using SELECT, the return code from the command or program is passed to the function that invoked SELECT.

**Note:** If a selected command, not using ISPF display services, could cause a full screen input or output operation, the developer should refresh the entire screen on the next display. To do this, use the CONTROL DISPLAY REFRESH service. See "Control Service" in Chapter 2.

  
page 122

The following return codes are possible if a menu is specified:

- 0 - Normal Completion. The END command was entered from the selected menu.
- 4 - Normal Completion. The RETURN command was entered or the EXIT option as specified from the selected menu or from some lower-level menu.
- 12 - The specified panel could not be found.
- 16 - Truncation error in storing the ZCMD or ZSEL variable.
- 20 - Severe error.

Example:

- In MVS or VM/SP (in a CLIST or EXEC), start a hierarchy of selection panels (menus) from a dialog function. The first menu in the hierarchy is named QOPTION.

```
ISPEXEC SELECT PANEL(QOPTION)
```

- In a PL/I program, program variable QOPT contains 'PANEL(QOPTION)' and program variable QOPTL is a full word containing the binary equivalent of 14. Start a hierarchy of selection panels (menus) beginning with panel QOPTION.

```
CALL ISPLINK('SELECT', QOPTL, QOPT);
```

- In MVS or VM/SP (in a CLIST or EXEC), invoke a program-coded dialog function named PROG1, and pass it a parameter string consisting of ABCDEF.

```
ISPEXEC SELECT PGM(PROG1) PARM(ABCDEF)
```

- In MVS or VM/SP, in a PL/I program, program variable PROG contains 'PGM(PROG1) PARM(ABCDEF)' and program variable PROGL is a full word containing the binary equivalent of 23. Invoke a program-coded dialog function, named PROG1, and pass it a parameter string consisting of ABCDEF.

```
CALL ISPLINK('SELECT', PROGL, PROG);
```

- In VSE, in a PL/I program, program variable PROG contains 'PGM(PROG1) PARM(ABCDEF) LANG(PLI)' and program variable PROGL is a full word containing the binary equivalent of 33. Invoke a program-coded dialog function (written in PL/I), named PROG1, and pass it a parameter string consisting of ABCDEF.

```
CALL ISPLINK('SELECT', PROGL, PROG);
```

*i.e. The ISPEXEC SELECT is itself coded in a CLIST or EXEC.*

## SETMSG - Set Next Message

The set next message service allows a dialog function to display a message on the next panel that is written by ISPF to the terminal. The next panel does not have to be displayed as a result of action taken by the function routine. The function routine may have, in fact, terminated before the next panel is displayed.

The specified message is retrieved from the message library at the time the set message request is issued. Values for all variables defined in the message are substituted at this time and the message is saved in a message area for the application. When the next panel is displayed, the message is retrieved from the save area and displayed on the panel.

If multiple set-message requests have been issued before a panel is displayed, only the last message is displayed. A message specified on the panel display request is overridden by any outstanding set next message request.

A message that has been set with SETMSG is displayed the next time any full screen output is sent to the display, regardless of whether that output is a panel, table display, browse data, or edit data.

The message is preserved across CONTROL NONDISPL. That is, the message is displayed on the next actual output to the terminal. If the next panel is processed in non-display mode, the message remains pending, to be displayed with any following panel that is processed in display mode.

If the message refers to a help panel, the help panel should not include substitutable variables. Variables in related help panel(s) contain the values current at the time the HELP command is issued, not at the time the SETMSG service is invoked.

Syntax for the set message function is:

---

```
ISPEXEC SETMSG MSG(message-id)
```

```
CALL ISPLINK ('SETMSG', message-id);
```

---

### **message-id**

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

The following return codes are possible:

0 - Normal completion.

12 - The specified message could not be found.

20 - Severe error.

Example: Put, on the next panel that is displayed, a message whose id, ABCX015, is in a dialog variable named TERMSG.

```
ISPEXEC SETMSG MSG(&TERMSG )
```

```
CALL ISPLINK('SETMSG', 'ABCX015 ');
```

## TBADD - Add a Row to a Table

The TBADD service adds a new row of variables to a table. The new row is added immediately following the current row, pointed to by the current row pointer (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 (that were specified by the KEYS and NAMES parameters in a TBCREATE) are saved in the row.

Additional variables (those not specified when the table was created) 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.

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.

---

```
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.

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 (zero).
- 12 - Table is not open.
- 20 - Severe error.



Example: Add a row to the table TELBOOK, copying to the row values from function pool variables whose names match those of table variables.

```
ISPEXEC TBADD TELBOOK
```

```
CALL ISPLINK('TBADD', 'TELBOOK ');
```

## TBBOTTOM - Set the Row Pointer to Bottom

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

All variables in the row, including keys, name, 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 a list of extension variable names contained in the row will be stored. The list must be enclosed in parentheses, and the names within the list must be separated by a blank.

The following return codes are possible:

- 0 - Normal completion.
- 8 - Table is empty; CRP set to TOP (zero).
- 12 - Table is not open.
- 16 - Variable value has been truncated or insufficient space provided to return all extension variable names.
- 20 - Severe error.

Example: Move the current row pointer (CRP) of the table TELBOOK to the last row of the table. Store values from variables in that row, in function pool variables having names that match the names of the variables in the row.

```
ISPEXEC TBBOTTOM TELBOOK
```

```
CALL ISPLINK('TBBOTTOM', 'TELBOOK ');
```

## TBCLOSE - Close and Save a Table

The TBCLOSE service terminates processing of the specified table and deletes the virtual storage copy, which is then 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.

Table output can be directed to a table output library other than the library specified on the table output ISPTABL DD, FILEDEF, or ISPDEF statement. (The library to be used must be allocated before table services receives control.) Thus, an application can update a specific table library. This is particularly useful for applications that need to maintain a common set of tables containing their data.

The output table library - specified by the ISPTABL DD, FILEDEF, or ISPDEF statement - is the default output library. Therefore, dialogs previously written for SPF that use table services continue to function in the same manner in ISPF as they did when running under SPF.

---

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

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

---

### **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 ensures 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 smaller than the table that replaces it, or if a member of the same name does not exist in the library, the complete table is written at the end of the output library.

**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.

## **alt-name**

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

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

The default value for this parameter is zero.

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 being updated in place.

## **library-name**

Specifies the name of a DD, FILEDEF, or ISPDEF statement that defines the output library in which table-name is to be closed. If this parameter is omitted, the default is ISPTABL.

The following return codes are possible:

- 0 - Normal completion.
- 12 - Table is not open.
- 16 - Table output library not allocated.
- 20 - Severe error.

Example: Close the table TELBOOK.

```
ISPEXEC TBCLOSE TELBOOK
```

```
CALL ISPLINK('TBCLOSE', 'TELBOOK ');
```

## 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.

---

```
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 the 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 non-key variables, by name, to be stored in each row of the table.

If key-name-list and name-list are omitted, the table can contain only extension variables that must be specified when the row is written.

#### WRITE

Specifies that the table is permanent, to be written to disk by the TBSAVE or TBCLOSE service. The disk copy is not actually created until the TBSAVE or TBCLOSE service is invoked.

#### NOWRITE

Specifies that the table is for temporary use only. When processing is complete, a temporary table should be deleted by the TBEND or TBCLOSE service.

#### 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 -- a duplicate table exists but REPLACE was specified.
- 8 - Table already exists; REPLACE was not specified.
- 12 - Table in use; ENQ failed.
- 16 - WRITE mode specified and table input library is not allocated. (TBCREATE checks the input library to determine if a duplicate table exists; see return code 8.)
- 20 - Severe error.

#### Example:

- In MVS or VM/SP, in a CLIST or EXEC, create a permanent table, TELBOOK, to contain the variable TABKEY and other variables, the names of which are specified in dialog variable TABVARS. The key field is TABKEY.

```
ISPEXEC TBCREATE TELBOOK KEYS(TABKEY) NAMES(&TABVARS )
```

- In a PL/I program, create a permanent table, TELBOOK, to contain the variable TABKEY and other variables, the names of which are specified in program variable TABVARS. The key field is TABKEY.

```
CALL ISPLINK('TBCREATE', 'TELBOOK ', 'TABKEY ', TABVARS);
```

## TBDELETE - Delete a Row from a 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 current row pointer (CRP) is deleted.

The CRP 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 from which the row is to be deleted. The row is determined by the current position of the CRP (if the table has no keys) or the current contents of the key variables (if the table has keys).

The following return codes are possible:

0 - Normal completion.

8 - Keyed tables: The row specified by the value in key variables does not exist; CRP set to TOP (zero).

Non-keyed tables: CRP was at TOP (zero) and remains at TOP.

12 - Table is not open.

20 - Severe error.

Example: Delete a row of the table TELBOOK.

```
ISPEXEC TBDELETE TELBOOK
```

```
CALL ISPLINK('TBDELETE', 'TELBOOK');
```



## TBDISPL - Display Table Information

The TBDISPL service combines information from a panel definition with information stored in an ISPF table. It displays all or selected rows from the table, allowing the application user to scroll the information up and down (only). An illustration appears in Appendix C, "Using the TBDISPL Service" on page 289.

When only selected rows from a table are to be displayed, the TBSARG service is used before issuing TBDISPL to define the selection criteria. In this case, ROWS(SCAN) must be specified on the )MODEL statement in the panel definition.

The format of the display is specified by a panel definition, which TBDISPL reads from the panel library. The panel definition contains two input fields (the command and scroll fields) and the non-scrollable text, which includes column headings. It also contains one or more "model" lines that specify the format of the scrollable data and which columns (i.e., which variables) from the table are to be displayed. For a description of panel formats for table display, see "Special Panel Requirements, Table Display Panels" in Chapter 7.

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 and may also enter commands in the command field.

Before TBDISPL is invoked, the table to be displayed must be open, and the current row pointer (CRP) positioned to the row in the table at which the display is to begin. (CRP at TOP is a value of 0 and is valid; it is treated as though the CRP were pointing to the first row.)

**Note:** Do not attempt to use TBDISPL to display a command table currently in use. The result would not be predictable.

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.

---

```
ISPEXEC TBDISPL table-name [PANEL(panel-name)]  
                               [MSG(message-id)]  
                               [CURSOR(field-name)]  
                               [CSRROW(table-row-number)]
```

```
CALL ISPLINK ('TBDISPL', table-name [,panel-name]  
                               [,message-id]  
                               [,field-name]  
                               [,table-row-number] );
```

---

**table-name**

Specifies the name of the table to be displayed.

**panel-name**

Specifies the name of the panel to be displayed.

**message-id**

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

**field-name**

specifies the column (variable name) where the cursor is to be positioned on the display. Any .CURSOR setting (done in the INIT section of the panel definition) takes precedence over this parameter.

**table-row-number**

specifies the table row number (CRP) corresponding to the line on the display where the cursor is to be positioned. If the specified row is not displayed on the screen, the cursor is placed at the command field. For call invocation, this parameter must be a full word fixed binary number.

The panel-name and message-id parameters are optional. Their processing is as follows:

- If panel-name is specified and message-id is not specified, the panel is read from the panel library, rows from the table are read to fill the screen, and the screen is displayed without a message.

- If panel-name and message-id are both specified, the panel is read from the panel library, rows from the table are read to fill the screen, and the screen is displayed with the specified message.
- If panel-name is not specified and message-id is specified, the current table display is overlaid with a message, without reinitializing the screen nor rereading the table.
- If neither panel-name or message-id is specified, the CRP is set to point to the table row corresponding to the next modified line on the display. If no modified lines remain to be processed, the following occurs:

If the application user's last act was to:

- Press the ENTER key, then rows from the table are again read to fill the screen and the screen is redisplayed.
- Enter a scroll command, then the scroll function is now honored by reading and displaying the appropriate rows from the table.
- Enter an END or RETURN command, then the CRP is set to TOP (zero) and return is made to the function (issuing the TBDISPL) with a return code of 8. If this occurs more than once in immediate sequence, a return code of 20 is issued, since the application may be in a loop.

The field-name and table-row-number parameters are optional. Their processing is as follows:

- If the field-name parameter is not specified but the table-row-number parameter is specified, the cursor is placed on the first field in the specified row.
- If the field-name parameter is specified but the table-row-number parameter is not specified, the current value of the CRP determines the row location and the cursor is placed in this row, on the field specified by the field-name parameter.
- Whenever the table-row-number parameter is specified, the contents of the row are always returned by TBDISPL, even if the user did not modify the row. This allows the dialog developer to force the user to correct an error on that row, before going on to process other rows.
- The .CURSOR control variable may be set within the )INIT section of a table display panel to specify a field name where within a row the cursor is to be initially positioned. If .CURSOR is not explicitly set, it assumes the value passed from the calling sequence in the field-name parameter, (if any). If cursor placement is not specified (in .CURSOR or a passed value), the cursor is positioned at the command input line.

TBDISPL allows the user to scroll the data up and down, and enter primary commands or information on one or more lines of scrollable data. After display of a panel using TBDISPL and after the user has made any modifications to the scrollable data on the display and has pressed the ENTER key or issued the SCROLL, END, or RETURN command, the TBDISPL service performs the following functions:

1. The contents of the command field are stored in the dialog variable specified in the panel definition.
2. If there was no modified line to process, the CRP is set to TOP (zero).
3. If the user entered information into one or more lines of scrollable data, the CRP is positioned to the row in the table that corresponds to the first modified line, and the row is retrieved (all variables from that row are stored into the corresponding dialog variables). The information entered by the user on that line is then stored in 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 in the system variable ZTDTOP. If, in a dialog, a dialog developer wants to reposition the scrollable data as the user last saw it, he must reposition the CRP to the row number stored in ZTDTOP prior to reinvoking the TBDISPL service.

(ZTDTOP is a variable in the function pool. A command procedure - CLIST or EXEC - may access it directly. A program may access it through use of the VCOPY service.)

5. TBDISPL then returns to the dialog function.

If the application user modified more than one line in a single interaction, a call to TBDISPL is required to position the CRP to the row corresponding to each modified line. After the CRP is positioned to each modified line, the function may process the line, for example, by issuing a TBPUT request to update the table. For these calls, neither the panel-name nor the message-id should be specified. The processing sequence for each of these calls is as described above, except that the next modified line is processed. The combination of the return code and CRP values indicate whether there are more modified lines to be processed.

Whenever modified table entries remain to be processed, the dialog can choose to ignore them by calling TBDISPL with a specified (non-blank) panel name. This clears out any remaining information about previous calls. If the dialog wants to display another table before processing all remaining entries from the first display, and then resume processing of multiple entries from the first display, it must invoke the CONTROL service to save and restore the display environment.

The following return codes are possible from TBDISPL:

- 0 - The ENTER key was pressed or a scroll command was entered. One or both of the following occurred:
  - one line was modified in the scrollable part of the display. The CRP is set to point to the table row corresponding to that line. The row is retrieved (i.e., stored in the function pool) and then the line is stored in the function pool.
  - a function command was entered by the user.
- 4 - The ENTER key was pressed or a scroll command was entered. The first or both of the following occurred:
  - two or more lines in the scrollable part of the display were modified by the user. The CRP is set to the table row corresponding to the first line changed. That row is retrieved (i.e., stored in the function pool) and then the line is stored in the function pool.
  - a function command was entered by the user.

For subsequent TBDISPL requests (with no panel name and no message-id), return code 4 is issued for each request until one modified line remains to be accessed. For this last line, a return code of zero is issued by the TBDISPL request (still specified with no panel name and no message-id).

- 8 - The END or RETURN command was entered. The CRP is set to the table row corresponding to the first of any lines modified in the scrollable part of the display. That row is retrieved (stored in the function pool) and then the line is stored in the function pool.

If the CRP is at the top (zero), no lines were changed.

To capture data entered when END or RETURN was entered, continue to issue TBDISPL requests with no panel name or message-id specified, until the CRP is at zero. For each request, a return code of 8 is issued.

In addition to the action described above, a function command may or may not have been originated by the user.

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

Example: (See Appendix C, "Using the TBDISPL Service" on page 289 for another example of TBDISPL use.)

Display the table TELBOOK using panel definition TPANEL2 to format the display.

```
ISPEXEC TBDISPL TELBOOK PANEL(TPANEL2)
```

```
CALL ISPLINK('TBDISPL', 'TELBOOK ', 'TPANEL2 ');
```

## TBEND - Close a 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.

Example: Delete the virtual storage copy table TELBOOK; do not change any permanent copy (in the table library).

```
ISPEXEC TBEND TELBOOK
```

```
CALL ISPLINK('TBEND', 'TELBOOK');
```

## TBERASE - Erase a Table

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

The table must not be open in WRITE mode when this service is invoked.

---

```
ISPEXEC TBERASE table-name [LIBRARY(library-name)]
```

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

---

### **table-name**

Specifies the name of the table to be erased.

### **library-name**

Specifies the name of a DD, FILEDEF, or ISPDEF statement that defines the library in which table-name exists. If this parameter is omitted, the default is ISPTABL.

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.

Example: Delete (erase) the table TELBOOK from the table library.

```
ISPEXEC TBERASE TELBOOK
```

```
CALL ISPLINK('TBERASE', 'TELBOOK');
```



## TBEXIST - Determine if a Row Exists in a 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 current row pointer (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; the CRP is positioned to the specified row.

8 - Keyed tables: the specified row does not exist; the CRP is set to TOP (zero).

Non-keyed tables: service not possible; the CRP is set to TOP.

12 - Table is not open.

20 - Severe error.

Example: In the keyed table TELBOOK, test for the existence of a row having a specific key value.

```
ISPEXEC TBEXIST TELBOOK
```

```
... if return code = 0 ...then the row exists...
```

```
CALL ISPLINK('TBEXIST', 'TELBOOK ');
```

```
... if return code = 0 ...then the row exists...
```

## TBGET - Retrieve a Row from a 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 current row pointer (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 a list of extension variable names contained in the row will be stored. The list is enclosed in parentheses, and the names within the list are separated by a blank.

The following return codes are possible:

0 - Normal completion.

8 - Keyed tables: The row specified by the value in the key variables does not exist; the CRP is set to TOP (zero).

Non-keyed tables: the CRP was at TOP and remains at TOP.

12 - Table is not open.

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

20 - Severe error.

Example: In the keyed table TELBOOK, for a row having a specific key value, copy values from variables in that row to variables in the function pool having names that match names of the variables in the row.

```
ISPEXEC TBGET TELBOOK
```

```
CALL ISPLINK('TBGET', 'TELBOOK ');
```

## TBMOD - Modify a Row in a 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 current row pointer (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 (keys and names) are saved in the row.

Additional variables, not specified when the table was created, may also be saved in the row. These "extension" variables apply only to this row; not the entire table. Whenever 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 the section entitled "Invocation of Services" for specification of name-lists.

The following return codes are possible:

- 0 - Normal completion. Keyed tables: Existing row updated. Non-keyed tables: New row added to table.
- 8 - Keyed tables: Keys did not match; new row added to the table.
- 12 - Table is not open.
- 20 - Severe error.

Example: Update or add a row of variables in the table TELBOOK using values from variables in the function variable pool.

```
ISPEXEC TBMOD TELBOOK
```

```
CALL ISPLINK('TBMOD', 'TELBOOK ');
```

## 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 ISPF. ISPTLIB may specify a concatenation of libraries. Refer to "Library Setup" in Chapter 4 for additional information.

An ENQ is issued to ensure that no other user is currently accessing the table. The ENQ applies only to the specified table (member) in the table input library - not to the entire library. For the WRITE option, it is an exclusive ENQ that remains in effect until the table is closed. For the NOWRITE option, it is a shared ENQ that remains in effect only during the time that the table is read into storage.

---

```
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 by use of the TBSAVE or TBCLOSE service. This option is the default.

### **NOWRITE**

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

The following return codes are possible:

- 0 - Normal completion.
- 8 - Table does not exist.
- 12 - ENQ failed; table in use by another user or the current user.
- 16 - Table input library not allocated.
- 20 - Severe error.

Example: Access (open) the table TELBOOK for updating.

```
ISPEXEC TBOPEN TELBOOK WRITE
```

```
CALL ISPLINK('TBOPEN', 'TELBOOK ', 'WRITE');
```

## TBPUT - Update a Row in a 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 current row pointer (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 variables not specified when the table was created, may also be saved in the row. These "extension" variables apply only to the row; not the entire table. Whenever 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 the section entitled "Invocation of Services" for specification of name lists.

The following return codes are possible:

0 - Normal completion.

8 - Keyed tables: The key does not match that of the current row; CRP set to TOP (zero).

Non-keyed tables: CRP was at TOP and remains at TOP.

12 - Table is not open.

20 - Severe error.



Example: Update a row, in the table TELBOOK, using values from variables in the function variable pool.

```
ISPEXEC TBPOT TELBOOK
```

```
CALL ISPLINK ('TBPOT', 'TELBOOK ');
```

## 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.

---

```
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 a list of key variable names contained in the table will be stored. 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 a list of variable names in the table, specified with the NAMES keyword when the table

was created, will be stored. 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 the number of rows contained in the table will be stored.

**keynum-name**

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

**namenum-name**

Specifies the name of a variable into which the number of variables in the table specified with the NAMES keyword when the table was created, will be stored.

**crp-name**

Specifies the name of a variable into which the current row pointer (CRP) number for the table will be stored. 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.

Example: For the keyed table TELBOOK:

- In dialog variable QKEYS, store the names of key variables
- In dialog variable QNAMES, store the names of non-key variables
- In dialog variable QROWS, store the number of rows

```
ISPEXEC TBQUERY TELBOOK KEYS(QKEYS) NAMES(QNAMES) ROWNUM(QROWS)
```

```
CALL ISPLINK('TBQUERY', 'TELBOOK ', 'QKEYS ', 'QNAMES ', 'QROWS ');
```

## TBSARG - Define a Search Argument

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

The search argument is specified in 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.

Generally, TBSARG is used prior to TBSCAN or TBDISPL operations to establish search arguments for these operations. To set up a search argument, set table variables in the function pool to nulls by using TBVCLEAR. Next, set a value in each variable that is to be part of the search argument. Then, issue TBSARG to establish this variable(s) as the search argument to be used in subsequently requested TBSCAN or TBDISPL operations.

Extension variables, only, may be included in the search argument by specifying their names in the name-list parameter. The values of these variables 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. This is called a generic search argument. A generic search argument is specified by placing an asterisk in the last non-blank position of the argument. Asterisks imbedded in the argument are treated as data. For example, to perform a generic search for a row value of DATA\*12, the generic search argument is:

```
DATA*12*
```

The first asterisk is part of the search argument. The second asterisk designates the argument to be a generic search argument.

**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 current row pointer (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 the 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.

Example: Establish a search argument to be used by a TBSCAN operation of the table TELBOOK. Assume that the argument values are contained in function pool variables whose names are the same as table variables in TELBOOK.

```
ISPEXEC TBSARG TELBOOK
```

```
CALL ISPLINK('TBSARG', 'TELBOOK ');
```

## TBSAVE - Save a 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.

Table output can be directed to a table output library other than the library specified on the table output ISPTABL DD, FILEDEF, or ISPDEF statement. (The library to be used must be allocated before table services receives control.) Thus, an application can update a specific table library. This is particularly useful for applications that need to maintain a common set of tables containing their data.

---

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

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

---

### **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.

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.

### **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 fullword fixed binary integer.

The default value for this parameter is zero.

### **library-name**

Specifies the name of a DD, FILEDEF, or ISPDEF statement that defines the output library in which table-name is to be saved. If this parameter is omitted, the default is ISPTABL.

The following return codes are possible:

- 0 - Normal completion.
- 12 - Table is not open.
- 16 - Table output library not allocated.
- 20 - Severe error.

Example: Write a table TELBOOK, previously opened and currently in virtual storage, to the table library. Retain the copy in virtual storage for further processing (do not close the table).

```
ISPEXEC TBSAVE TELBOOK
```

```
CALL ISPLINK('TBSAVE', 'TELBOOK ');
```



## TBSCAN - Search A Table

The TBSCAN service searches a table for a row with values that match an argument list. The argument list may be established by use of 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 current row pointer (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 become part of the search argument. A value of the form AAA\* means that only the characters up to the "\*" are compared. This is called a generic search argument. A generic search argument is specified by placing an asterisk in the last non-blank position of the argument. Asterisks imbedded in the argument are treated as data. For example, to perform a generic search for a row value of DATA\*12, the generic search argument is:

```
DATA*12*
```

The first asterisk is part of the search argument. The second asterisk designates the argument to be a generic search argument.

**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(*)
```

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 the section entitled "Invocation of Services" for specification of name lists.

**var-name**

Specifies the name of a variable into which a list of extension variable names contained in the row will be stored. 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 (zero).

12 - Table is not open.

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

20 - Severe error.

Example: For the table TELBOOK:

- Establish a search argument to be used by a TBSCAN operation of table TELBOOK.

```
ISPEXEC TBSARG TELBOOK
```

```
CALL ISPLINK('TBSARG', 'TELBOOK ');
```

- Move table TELBOOK's CRP to the row that fulfills the search argument as specified in the TBSARG operation, above. Copy values from variables in that row to function pool variables whose names match those of the table variables.

```
ISPEXEC TBSCAN TELBOOK
```

```
CALL ISPLINK('TBSCAN', 'TELBOOK ');
```

If the return code is 0, the row was found and values were copied from the row variables to function pool variables.

## TBSKIP - Move the Row Pointer

The TBSKIP service moves the current row pointer (CRP) of a 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 fullword fixed binary number.

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

### **var-name**

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

The following return codes are possible:

- 0 - Normal completion.
- 8 - CRP would have gone beyond the number of rows in the table (this includes a table empty condition); CRP set to TOP (zero).
- 12 - Table is not open.
- 16 - Variable value has been truncated or insufficient space provided to return all extension variable names.
- 20 - Severe error.

Example: In the table TELBOOK, move the current row pointer (CRP) to the next row. After the move, copy values from variables in that row to variables in the function variable pool having names that are the same as the names of the variables in the row.

```
ISPEXEC TBSKIP TELBOOK
```

```
CALL ISPLINK('TBSKIP', 'TELBOOK');
```

## TBTOP - Set the Row Pointer to the Top

The TBTOP service sets the current row pointer (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.

Example: For the table TELBOOK, move the current row pointer (CRP) to the row immediately before its first row.

```
ISPEXEC TBTOP TELBOOK
```

```
CALL ISPLINK('TBTOP', 'TELBOOK ');
```

## 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.

The contents of the table and the position of the current row pointer (CRP) are not changed by this service.

---

```
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.

Example: Clear dialog variables associated with the table TELBOOK to nulls.

```
ISPEXEC TBVCLEAR TELBOOK
```

```
CALL ISPLINK('TBVCLEAR', 'TELBOOK ');
```

ISPF → Ad

## VCOPY - Create a Copy of a Variable

This service is used only with call invocations.

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

The variable names may be specified as a single 8-character value, a list enclosed in parentheses, or a name-list structure. In LOCATE mode an array of pointers must be supplied to receive the data address. An array of lengths must be supplied to receive the data lengths.

The VCOPY service automatically allocates storage for the data, and returns the address and length of each variable to the caller. In MOVE mode, an array of lengths must be supplied on input; its values map the structured area which must be supplied to receive the data. 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. The length array is then set with the data lengths.

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

---

ISPEXEC \*This service does not apply to command procedures\*

```
CALL ISPLINK ('VCOPY', name-list, length-array, value-array
              [, 'LOCATE' | 'MOVE' ] );
```

---

### name-list

Specifies an area containing the names of dialog variables to be copied. The standard name-list format is used.

### length-array

Specifies an array of fullword fields containing the lengths of the data areas for the dialog variable values. (The array may consist of a single item.) In move mode, each element of the array is set by the caller to the output area size. In move mode or locate mode, each element of the array is set by the service to the number of bytes of data for the corresponding variable.

11 or 12

### value-array

In locate mode, specifies the name of an array that contains pointers to fields into which the copied variables are placed. (The array may consist of a single item.)

In move mode, specifies the name of a structure that is mapped by the length array.

### LOCATE

Specifies locate mode. The address of the copied value is returned to the user invoking the service.

### MOVE

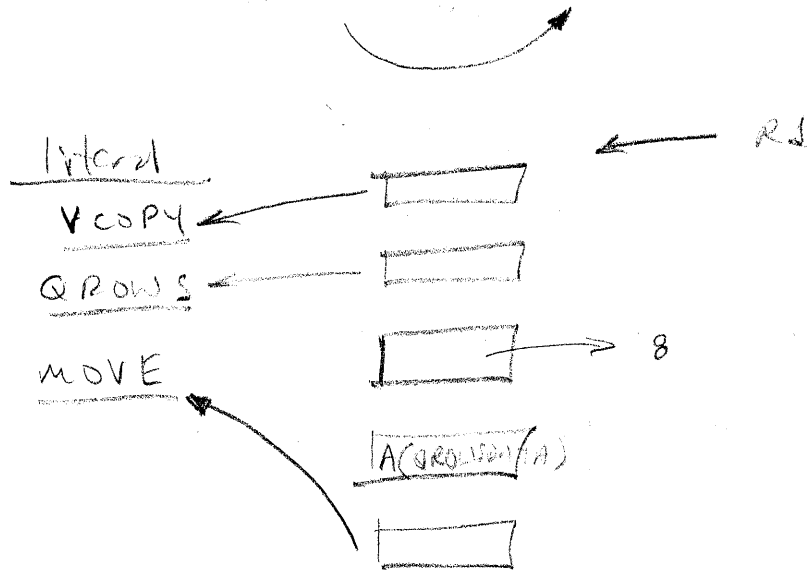
Specifies move mode. The copied value is returned to the user invoking the service.

The following return codes are possible:

- 0 - Normal completion.
- 8 - One or more variables do not exist.
- 16 - Truncation has occurred during data movement (move mode only).
- 20 - Severe error.

Example: Copy the value in dialog variable QROWS to a field named QROWSDATA in this PL/I program module. Perform the copy in move mode (as opposed to locate mode). Variable L8 contains a value of 8.

```
CALL ISPLINK('VCOPY', 'QROWS ', L8, QROWSDATA, 'MOVE');
```





## VDEFINE - Define Function Variables

The VDEFINE service is used only with call invocations.

The VDEFINE service is invoked by a program to give ISPF the ability to use dialog variable names to directly access variables within the particular program module. In the call to VDEFINE, the program specifies the format (character string, fixed binary, bit string, hex or user-defined) and length of the variables. (As described in "The Function Pool for Programs" in Chapter 2, stacking of definitions for a particular dialog variable may be achieved by successive calls to VDEFINE for that dialog variable.)

When the VDEFINE service is called, ISPF enters the dialog variable names into the function pool for the dialog function currently in control. Dialog variables entered in the function pool by use of the VDEFINE service are called defined variables to distinguish them from implicit variables in the function pool. (Refer to "The Function Pool for Programs" in Chapter 2, for a further discussion of defined and implicit variables.)

A list of dialog variables can be defined with a single call to the VDEFINE service. The program variables that correspond to the dialog variables defined to ISPF by VDEFINE must have the same format and length and be in contiguous locations or defined as an array or structure within the program. The program variable name passed to ISPF must be the name of the first variable as defined in the program, the name of the array, or the name of the structure. The format and length specified in the call to VDEFINE are the format and length of each individual variable.

EXIT ROUTINE: The dialog writer may specify an exit routine to define dialog variables when program variables are in non-standard formats (formats other than CHAR, FIXED, BIT, or HEX). When a variable is then accessed by any ISPF service, the exit routine is invoked to perform any conversion necessary between the program variable's format and the character string format required for a dialog variable. Appendix F, "VDEFINE Exit Routine," describes invocation, parameters used, and return codes required for the exit routine.

---

ISPEXEC \*This service does not apply to command procedures\*

```
CALL ISPLINK ('VDEFINE', name-list, variable, format, length
             [,options-list][,user-data]);
```

---

### name-list

Specifies the symbolic name or name-list to be used by ISPF when referencing the specified variables.

An asterisk specifies (in conjunction with the USER format keyword described below) that the exit routine (whose address is specified in the user-data parameter) is to be called for variables not found in the function pool.

### **variable**

Specifies the variable whose storage is to be used. If a name list is passed, this storage contains an array of variables. The number of names in the list determines the dimension of the array.

### **format**

Specifies the data conversion format.

The format parameters are:

#### **CHAR**

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

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

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

#### **FIXED**

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

Fixed variables that have a length of 4 bytes (fullword) are treated as signed, represented 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 of 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 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 by the characters 0-9 and A-F. Within the variable, the data is left-justified and padded on the right with binary zeros.

#### **USER**

Specifies that the format is to be determined by the user. Any conversion format is allowed. A conversion routine must

be specified and is specified by naming it in the user-data parameter, described below.

### **length**

Specifies the length of the variable storage, in bytes. This parameter must be a full word binary integer. The maximum length for a FIXED variable is 4 bytes. The maximum length for other types of variables is 32,767 bytes.

### **options-list**

Specifies initialization of the defined storage and/or retention of trailing blanks in variable data.

The options-list parameters are COPY and NOBSCAN (both may be specified and are specified in the name-list format):

#### **COPY**

Specifies that any dialog variable with the same name may be used to initialize the defined storage. The variable pools are searched in the standard (function pool, shared pool, profile pool) sequence.

#### **NOBSCAN**

Specifies that any trailing blanks in the variables are to remain in the variables.

### **user-data**

Specifies the storage location that contains the entry point address of the conversion subroutine followed by any other data that should be passed to the routine. This parameter is specified whenever the USER parameter is specified.

The following return codes are possible:

- 0 - Normal completion.
- 8 - Variable not found.
- 16 - Data truncation occurred.
- 20 - Severe error.

Example: Establish ISPF accessibility, using the name ZMSGNAME, to a field named ZERRMSG in this PL/I module. The field is a character string and is 8 bytes long. Program variable L8 contains a value of 8.

```
CALL ISPLINK('VDEFINE', 'ZMSGNAME', ZERRMSG, 'CHAR', L8);
```

## VDELETE - Remove a Definition of Function Variables

The VDELETE service is used only with the call invocations.

The VDELETE service removes variable names, previously defined by the program module, from the function pool. This service is the opposite of VDEFINE.

---

ISPEXEC \*This service does not apply to command procedures\*

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

---

### **name-list**

Specifies the dialog variable names that are to be removed from the function pool, or contains an asterisk.

An asterisk (\*) specifies removal, from the function pool, of all dialog variable names previously defined by the program module, including exit routine definitions.

The following return codes are possible:

- 0 - Normal completion.
- 8 - At least one variable not found.
- 16 - Data truncation occurred.
- 20 - Severe error.

Example: Remove ISPF accessibility to PL/I program variable ZERRMSG that was previously established by VDEFINE to be accessible using dialog variable name ZMSGNAME.

```
CALL ISPLINK ('VDELETE', 'ZMSGNAME');
```

## VGET - Retrieve Variables from a Pool or Profile

The VGET service copies values from dialog variable(s) in the shared variable pool or the application profile to the function pool variables with the same names. If a named function variable already exists, it is updated; if not, it is created and then updated.

---

```
ISPEXEC VGET name-list [ASIS|SHARED|PROFILE]
```

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

---

### name-list

Specifies the names of one or more dialog variables whose values are to be copied from the shared or profile pool to the function pool. The names are passed in the standard name-list format. See "Invocation of Services" for specification of name lists.

P.104  
↗  
↘

### ASIS

Specifies that the variables are to be copied from the shared variable pool, or, if not found there, from the profile pool.

### 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 application profile. A shared pool variable with the same name is deleted, even if it is not found in the profile pool.

The following return codes are possible:

- 0 - Normal completion.
- 8 - Variable not found.
- 16 - Translation error or truncation occurred during data movement.
- 20 - Severe error.

Example: In MVS or VM/SP, in a CLIST or EXEC, copy from the shared pool to the function pool, values for variables whose names are listed in variable VARLIST.

```
ISPEXEC VGET (&VARLIST ) SHARED
```

In a PL/I program, VARLIST contains a list of variable names. Copy values for these variables, from the shared pool to the function pool.

```
ISPLINK('VGET' VARLIST 'SHARED');
```

## VPUT - Update Variables in a Pool or Profile

The VPUT service copies values from dialog variables in the function pool to the shared pool or to the application profile. If a variable of the same name already exists in the shared pool or the profile, it is updated. If it does not exist, it is created and then it is updated.

---

```
ISPEXEC VPUT name-list [ASIS|SHARED|PROFILE]
```

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

---

### name-list

Specifies the names of one or more dialog variables whose values are to be copied from the function pool to the shared or profile pool. See "Invocation of Services" for specification of name lists.

### ASIS

Specifies that the variables are to be copied to the pool that they already exist in or that they are to be copied to the shared pool, if they are new. If the variables exist in both the shared and profile pools, they are copied to the shared pool only.

### 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 application profile. Any shared pool variable(s) of the same name(s) are deleted.

The following return codes are possible:

- 0 - Normal completion.
- 8 - Variable not found.
- 16 - Truncation has occurred while copying variables to the application profile.
- 20 - Severe error.

Example: In MVS or VM/SP, in a CLIST or EXEC, write variables, the names of which are listed in the variable VPUTLIST, from the function variable pool to the shared variable pool.

```
ISPEXEC VPUT (&VPUTLIST ) SHARED
```

*required blank*

In a PL/I program, write variables, the names of which are listed in program variable VPUTLIST, from the function variable pool to the shared variable pool.

```
ISPLINK ('VPUT' VPUTLIST 'SHARED ');
```



Ad2 → ISPF

## VREPLACE - Replace a Variable

The VREPLACE service is used only with call invocations.

The VREPLACE service allows a program module to update the contents of a variable in the function pool.

The variable names may be specified as single 8-character values, a list enclosed in parentheses, or a name-list structure. An array of lengths must be supplied on input to map the area that contains the data for each of the variables.

The variable to be updated can be 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 is created as an implicit function variable.

---

ISPEXEC \*This service does not apply to command procedures\*

CALL ISPLINK ('VREPLACE', name-list, lengths, values);

---

### **name-list**

Specifies the names of the dialog variables whose values are to be updated. The standard name-list format is used.

### **lengths**

Specifies an array of values giving, for each corresponding variable in the name-list, the number of bytes of the data to be used in the updating. Each field in the array must be a fullword binary integer.

### **values**

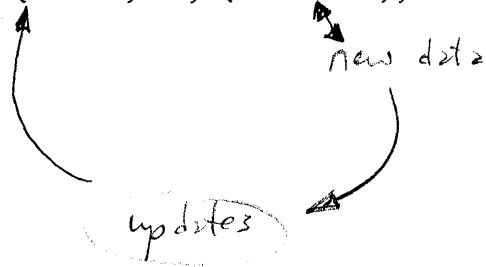
Specifies (in the buffer mapped by the length array) the update data to be used in the updating.

The following return codes are possible:

- 0 - Normal completion.
- 16 - Truncation has occurred during data movement.
- 20 - Severe error.

Example: Copy the value of a field named QROWSDATAZ from this PL/I program module to the function variable named QROWS. Before the copy operation, if no variable with this name is found in the function pool, create one giving it the name QROWS. Program variable L8 contains a value of 8.

```
CALL ISPLINK('VREPLACE', 'QROWS ', L8, QROWSDATAZ);
```



## VRESET - Reset Function Variables

The VRESET service is used only with call invocations.

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

Any defined variables are removed from the function pool (as though VDELETES had been done); any implicit variables are also deleted.

---

ISPEXEC \*This service does not apply to command procedures\*

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

---

The following return codes are possible:

0 - Normal completion.

20 - Severe error.

Example: Remove ISPF accessibility to all PL/I program variables.

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

## CHAPTER 7. PANEL AND MESSAGE DEFINITION AND SKELETON FORMATS

This chapter describes the syntax for defining panels, messages, and file tailoring skeletons. If ISPF/PDF is installed, consider using its modeling facilities to assist in entering panel definitions (see ISPF/PDF Reference).

### PANEL DEFINITIONS

ISPF panel definitions are stored in a panel library and are displayed by means of the SELECT, DISPLAY, or TBDISPL service. Each panel definition is referenced by its 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 are used in the body of the panel definition to represent attribute (start of field) bytes. Default attribute characters are provided, but 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. Model section (table display panels only) - defines the format of each line of scrollable data. This section is required for table display panels, and is invalid for other types of panels.
4. Initialization section (optional) - specifies the initial processing that is to occur prior to displaying the panel. This section is 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. This section is typically used to define how variables are to be verified and translated.

The panel definition is always ended with an )END statement, regardless of the sections included in the definition.

The sections must appear in the order listed above. The sections are delineated by the following statements:

```
)ATTR - start of attribute section
)BODY - start of body section
)MODEL - start of model section (applies to table display panels only)
)INIT - start of initialization section
)PROC - start of processing section
)END - end of panel definition
```

The discussion of panels is arranged as follows:

- The formatting guidelines, which indicate the placement of panel elements.
- A discussion of each section of the panel, in the order in which they appear on the panel. (The model section is described with table display panels, since it is unique to that type of panel.)
- A discussion of how panel processing is done.
- The rules and restrictions regarding the syntax of statements, variable names, and keywords.
- A discussion of special panel types: menus, help/tutorials, and table displays.

## Formatting Guidelines

This section presents general guidelines for panel design.

Generally, in any panel definition, the first three displayable lines include system-defined (default) areas for the display of messages, a command/option field, and a scroll field. Location of the message areas and command field may be overridden, see "Panel Body Section."

Following are suggestions for formatting the first three lines of a panel body:

|        |                |               |
|--------|----------------|---------------|
| line 1 | Title          | Short Message |
| line 2 | Command/Option | 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 in the default short message area.

If short messages are used, they should provide a brief indication of:

- Successful completion of a processing function, or
- Error conditions (accompanied by audible alarm).

Short messages temporarily overlay whatever information is currently displayed in the right-hand end of the first line, and are removed from display on the next interaction. (The original information is redisplayed when the message is removed.)

Short messages should be used either consistently throughout the application, or not at all.

For table display, the short message area 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.

Line 2 generally contains the command field. This same field should be used for option entry on menus. The command field, when the first input field on the panel or when identified by using keyword CMD in the panel body section, may be named using any valid variable name. However, the name ZCMD is generally used.

(Cursor placement while viewing a panel differs between use of the name ZCMD and other names. When ZCMD is used and cursor placement is not explicitly specified, ISPF will skip over the command field - when blank - to place the cursor on a following input field. When a name other than ZCMD is used, a blank command field is not skipped over when placing the cursor during display.)

For table display, edit, and browse panels, the scroll field should be at the right-hand end of line 2, following the command/option area. The scroll field must be the second input field in the panel definition, and must be four characters in length. A scroll field is not meaningful for other types of panels, and should be omitted from them.

Line 3 should generally be left blank, so that long messages do 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. Refer to "Panel Body Section" for a discussion of how to specify the information in the first three lines of the panel.

**Note:** The command input field should not be located on line 3, the line on which long messages display, because when display of a long message is superimposed on the command line, results are unpredictable.

Following are additional suggestions for designing panels:

- 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.

**Note:** Place the command line near the top of the panel. If the command line is near the bottom, and the end user enters split screen mode, the command line may not be visible on the screen and, therefore, the user may not be able to continue processing the dialog.

- 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. As examples:

FILE NAME ==> (arrow signals an input field)

EMPLOYEE SERIAL: 123456 (colon signals a protected field)

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.

## Syntax Rules and Restrictions

This section describes the syntax to be used in panel definition statements.

## General Rules

The following general syntax rules apply:

- 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.
- Input and output fields defined in the body section may not exceed 255 bytes in length.
- All header statements, such as )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 is also 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 by using a keyword on the )BODY statement.
- An )END statement is required as the last line of each panel definition.

## Blanks and Comments

The following rules apply to the use of blanks and comment statements:

- 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 )
```



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 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.
- In MVS and VM/SP, blank lines may occur anywhere within the attribute, initialization, and processing sections.
- In MVS, VM/SP, and VSE/AF 2.1, you can use blank lines in a panel definition. In VSE/AF 1.3.5, however, blank lines **cannot** be used in a panel definition. For VSE/AF 1.3.5, you can cause a blank line to appear in a panel display by providing, in the panel definition, a line that is blank except for a field delimiter character, such as "+", in the first position.

## Lists

The following rules apply to items in lists:

- Listed items within parentheses may be separated by commas or one or more blanks. This includes paired values within a TRANS statement. For example, the following are 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 as blank items. For example, the following are equivalent:

```
TRANS (&XXX N,' ', Y,YES, *,' ')
TRANS (&XXX N,, Y,YES, *,)
```

- Listed items within parentheses may be continued on one or more lines. For 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 Literal Expressions

The following rules apply to variables in 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 all three sections, 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'
```

- A period (.) at the end of a variable name causes concatenation with the character string following the variable. For example, if &V has the value ABC, then

```
'&V.LMN' yields 'ABCLMN'
```

- 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. For example:

```
%DATA SET NAME: &DSNAME -----%
```

Assuming that the value of variable DSNAME is greater than seven characters, the dashes are "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 are "pulled" in from the right.

*Examples  
P. 249  
P. 295*

## Attribute Section

The attribute section of a panel defines the special characters that are to be used in the definition of the body of the panel 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.

If specified, the attribute section precedes the panel body. It begins with the )ATTR header statement and ends with the )BODY header statement.

### Default Attribute Characters

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 which consist 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.

The maximum length of dialog variables, used for input or output in a panel, is 255 bytes.

The default characters may be changed by means of a keyword on either the )ATTR or )BODY header statement. For example:

```
DEFAULT(abc)
```

where "a", "b", and "c" are the three characters that 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. For example:

```
)BODY DEFAULT($ç_)
```

In this example, the default characters for text fields are changed to "\$" for high intensity, and "ç" for low intensity. The default character for high intensity input fields is specified as "\_" (the same as from the ISPF-supplied default).

### Statement Formats

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.

Generally, 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: TYPE, INTENS, CAPS, JUST, PAD, SKIP, and ATTN. They are all optional, except that at least one parameter must be specified. They may be specified in any order.

In attribute keywords, the "value" may be expressed as a literal or as a dialog variable name, preceded by an ampersand (&). For 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; therefore,

```
TYPE(&A)
```

is **invalid** even when the current value of dialog variable A is TEXT.

For simplicity, the values in the following examples are shown as literals.

### TYPE(TEXT|INPUT|OUTPUT)

Specifies the type of the field:

- TEXT - text (protected) field
- INPUT - input (unprotected) field
- OUTPUT - output (protected) field

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 there is no default attribute character for output fields.

**INTENS(HIGH|LOW|NON)** Specifies the intensity of field:

- HIGH - high intensity field
- LOW - low (normal) intensity field
- NON - non-display field (valid only for input fields) — *masked*

### CAPS(ON|OFF)

Specifies the upper or lower case attribute of the field, and is valid only for input and output fields:

- ON - translate to uppercase
- OFF - no translation

For CAPS ON, initial values and values entered by the user are automatically translated to uppercase. For CAPS OFF, no translation is performed.

**Note:** Use of CAPS (OFF) is negated if the dialog variable is referenced in a CLIST. (CLISTs translate unconditionally to upper case).

### JUST(LEFT|RIGHT|ASIS)

Specifies how the contents of the field are to be justified, and is valid only for input and output fields.

LEFT - left justification

RIGHT - right justification

ASIS - no justification

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(NULLS|char)**

Specifies the pad character for initializing the field, and is valid only for input and output fields.

NULLS - nulls are used for padding.

char - any character, including blank ( ' ') may be specified as the padding character.

If this keyword parameter is omitted, the default is user-defined for input fields and blank for output fields. The user-defined default for input fields is specified by using ISPF Parms (see Appendix B, "Using the ISPF PARMS Option.")

If the field is initialized to blanks (or the corresponding dialog variable is null), the entire field contains the pad character when the panel is first displayed. If the field is initialized with a value, the remaining field positions (if any) contains 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 extends to the right. For right justified fields, the padding extends to the left.

The pad characters are automatically deleted when the field is processed.

When an input field is processed, leading or trailing pad characters are automatically deleted, as follows:

- For a left justified field, leading and trailing pad characters are deleted.

- For a right justified field, leading pad characters are deleted.
- For an ASIS field, trailing pad characters are deleted.

In no case are imbedded pad characters deleted; only leading or trailing pad characters are deleted.

In the PAD(NULLS|char parameter is omitted, the default for input fields is either nulls or blanks, as specified by a user by use of the terminal characteristics panel (option 0.1); for output fields the default is blanks.

### SKIP(ON|OFF)

The SKIP keyword defines the autoskip attribute of the field, and is valid only for text or output (protected) fields.

SKIP(ON) specifies that the cursor automatically skips the field and is positioned to the first character location of the next unprotected field, upon entry of a character into the last character location of the preceding unprotected data field.

SKIP(OFF) specifies that the cursor does not automatically skip the field when the above condition occurs.

### ATTN(ON|OFF)

The ATTN keyword defines the attention-select attribute of the field, and is valid only for text fields.

ATTN(ON), specifies that the field may be selected by using the light pen or cursor select key (see "Light Pen and Cursor Select" in Chapter 3, "Use of Commands, Program Keys, and Light Pen").

ATTN(OFF), specifies that the field is not sensitized for selection in this manner.

**Note:** The panel designer must provide an adequate number of blank characters before and after the attention attribute character, as required by the 3270 hardware (see 3270 Information Display System Component Description, GA27-2749).

## Panel Body Section

The body 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.

Several keywords may optionally be placed on the )BODY header statement. They include KANA, CMD, SMSG and LMSG. The KANA keyword should be included when Katakana characters are to appear within the panel (see Appendix G, "Character Translations for APL, TEXT, and Katakana").

### Command and Message Fields

There are system-defined (default) areas for the display of messages and the command field (see "Panel Definitions"). Alternate locations may be specified by use of the following keywords on the )BODY header statement:

- CMD(field-name) - identifies the panel field (variable name) that is to be treated as the command field. The field must be TYPE(INPUT).

**Note:** If a command line is omitted from a panel, the first input field is used by ISPF as a default command field.

- SMSG(field-name) - identifies the panel field (variable name) where the short message, if any, is to be placed. The field must be TYPE(OUTPUT). If the length of this field is shorter or longer than 24 characters, the message is truncated or justified as appropriate.
- LMSG(field-name) - identifies the panel field (variable name) where the long message, if any, is to be placed. The field must be TYPE(OUTPUT). If the length of this field is shorter or longer than 79 characters, the message is truncated or justified as appropriate.

### Sample Body Section

The sample panel definition, shown in Figure 24 consists of a panel body followed by an ")END" control statement. It has no attribute, initialization, or processing sections, and uses the default attribute characters.

This data entry panel has 11 input fields (ZCMD, TYPECHG, LNAME, etc.), indicated with underscores. It also has a substitutable variable (EMP SER) within a text field (on line 2). The first two lines of the panel and the arrows preceding the input fields are all highlighted, as indicated by the percent signs. The other text fields are low intensity, as indicated by the plus signs.

Following )INIT section processing and immediately before a panel is displayed, all variables in the panel body are automatically initialized from the corresponding dialog variables (EMP SER, TYPECHG, LNAME, etc.). After the panel has been displayed and before )PROC section processing, the input fields are automatically stored into the corresponding dialog variables.

Figure 25 shows the panel as it appears when displayed, assuming that the current value of EMP SER is "123456", and that the other variables are initially null.



```

)BODY
%----- EMPLOYEE RECORDS -----
%COMMAND ==> _ZCMD +
%
%EMPLOYEE SERIAL: &EMP SER
+
+ 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 24. Sample Panel Definition

---

```
----- EMPLOYEE RECORDS -----
COMMAND ==>

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 25. Sample Panel - When Displayed

---

## Model Section

The model section is used only with table display panels and defines how each table row is to be formatted. Because the model section is unique to table display panels, it is discussed in "Table Display Panels."

↪ p. 243

## Initialization and Processing Sections

The initialization and processing sections are discussed together because the same statements may be used in both, although typically certain statements are used only in the initialization section while others appear only in the processing section.

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 variables that are displayed in the panel body reflect the contents of the corresponding dialog variables after the )INIT section has been processed, just prior to display of the panel. The input fields are automatically stored into the corresponding dialog variables immediately following display and prior to processing the )PROC section.

The processing section specifies 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.

### Statement Formats

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

There are four types of statements that may be used in these sections: assignment, IF, VER (verify), and VPUT. 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 the 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 "Syntax Rules and Restrictions."

In the description of statements and built-in functions that follows, a "variable" may be either a dialog variable or a 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. For example:

```
&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. The literal ' ' represents a single blank. To define a null, you must use the &Z literal.

#### TRUNC (variable,value)

p. 205

This built-in function may occur on the right side of an assignment statement to cause truncation. The first parameter inside the parentheses specifies the variable to be truncated. The value 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. For example:

```
&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 contains "3".

The control variable .TRAIL contains the "remainder" following a TRUNC operation. When the contents of a variable are truncated to a specified length, all remaining characters are stored in .TRAIL. If the contents of a variable are truncated at the first occurrence of a special character, the remaining characters following the special character are stored in .TRAIL. The special character, itself, is not stored nor is it retained in ZCMD, the command field. For example:

```
)PROC  
  &AAA = TRUNC (&ZCMD, '.')  
  &BBB = .TRAIL
```

If variable ZCMD contains "9.4.6", variable AAA contains "9" and the .TRAIL control variable and the variable BBB contains "4.6".

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

This built-in function may occur on the right 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. For 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. For example:

```
&REPL = TRANS (&MOD ... *,'?')  
&REPL = TRANS (&MOD ... *,*) } → see p. 229
```

In the first example, if the current value of MOD does not match any of the listed values, a question mark is 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 is 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. For 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 26 shows an example in which it is assumed that variable TYPECHG was originally set (in the dialog function) to a single character "N", "U", or "D". In the )INIT section, TYPECHG 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 TYPECHG by simply overtyping the first character.

Finally, the TRANS and TRUNC built-in functions may be nested. For example:

```
&XYZ = TRUNC( TRANS(&A ---),1 )
&ZSEL = TRANS( TRUNC(&ZCMD,'.') --- )
```

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 ZCMD are truncated at the first period, the truncated value is then translated, and the result is stored in variable ZSEL.

---

```
)BODY
%----- EMPLOYEE RECORDS -----
%COMMAND====>_ZCMD +
+
%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 26. 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. For example:

```
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 the value of 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, then 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 stated condition is false.

Figure 27 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.

```

)BODY
%----- EMPLOYEE RECORDS -----
%COMMAND====>_ZCMD
+
+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 27. Sample Panel with IF Statement

### VER (variable [,NONBLANK], 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 NONBLANK, a keyword specifying that the variable is to contain a value and not a blank. 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 in the MSG=value parameter, where "value" is a message-id. If no message is specified, an ISPF-supplied message is displayed, based on the type of verification. Even if a VER fails, processing of the panel INIT and PROC sections is performed.

ISPF provides several types of verification, as 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 and must not be blank. NONBLANK (or the abbreviated form, NB) may be specified with another type verification (ALPHA, NUM, HEX, etc.) by specifying the NONBLANK keyword first (after the variable name but before the other keyword). Example:

```
VER (&A,NB,PICT,NNN-NNNN)
```

is equivalent to:

```
VER (&A,NONBLANK)
VER (&A,PICT,NNN-NNNN)
```

- 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,BIT) - The variable must contain all '0's and '1's.
- VER (xxx,FILEID) - The variable must contain a valid fileid (in CMS syntax), which is any value that is valid with a LISTFILE command. The file name and file type (if given) must be one to eight alphanumeric characters (A-Z, 0-9, #, \$, @, the filemode must be a single letter (A-Z), optionally followed by a single digit (0-9). In addition, one or more fields of the file identifier may be an asterisk (\*) or may be a string of characters followed by an asterisk (e.g., tr\*).
- 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 hexadecimal 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.

**Note:** The length of the variable value and the picture string must be the same. Trailing blanks are not included.

- 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 TSO data set name. A data set name qualifier must begin with an alphabetic character, @, #, \$, or a period. The remaining characters must be either alphameric or a hyphen (-).

**Note:** ISPF Dialog Manager does not take the length of the actual TSO prefix into account when the panel user specifies a data set name without quotes. It checks that the length of the variable is no greater than 42 characters.

- VER (xxx,RANGE,lower,upper) - The variable must be numeric, and its value must fall (inclusively) within the specified lower and upper limits.

**Note:** The length of the specified variable is limited to 16 digits. Further, the lower and upper parameters may consist of no more than 16 digits each.

- 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 28 shows a sample panel with VER statements to verify that information entered by the user meets the following criteria:

- The truncated value of TYPECHG is "N", "U", or "D".
- The three name variables (LNAME, FNAME, and I) contain all alphabetic characters.
- The PHA (area code) field contains all numeric characters.

- The PHNUM (local number) field 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 ISPF-provided message will be displayed if the variable fails the verification test.

```

)BODY
%----- EMPLOYEE RECORDS -----
%COMMAND====>_ZCMD
+
%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 28. Sample Panel with Verification

## VPUT name-list [ASIS|SHARED|PROFILE]

While variables entered from a panel are automatically stored in the function variable pool, variables can be stored in the shared and profile variable pools by VPUT statements used in the )INIT or )PROC sections of the panel definition. For example:

```
)PROC
  VPUT (XYZ ABC) PROFILE
```

This coding causes current values for variables XYZ and ABC to be stored in the profile pool by a VPUT operation.

The syntax for the VPUT statement is the same as that for the VPUT service when it is invoked from a CLIST or EXEC 2 except that the ISPEXEC command verb is omitted. (Refer to Chapter 6, "Description of Services" for a description of the VPUT parameters).

### Control Variables

Control variables are used to control and test certain conditions pertaining to the display of a panel. The control variables are described below:

*p. 226* ↙  
**.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. For example:

```
.CURSOR = DSN
```

causes the cursor to be placed at the beginning of field DSN. This variable is automatically set to the field last referenced whenever .MSG is set explicitly or indirectly by TRANS or VER statements.

**.HELP** May be set in the initialization section to establish a tutorial (help) panel to be displayed if the user enters the HELP command. For example:

```
.HELP = ISPTE
```

causes tutorial page ISPTE to be displayed when the user enters the HELP command.

**.MSG** May be set to a message-id, typically in the processing section, to cause a message to be displayed. For example:

```
.MSG = ISPE016
```

This variable is automatically set by use of the MSG=value keyword on a TRANS (if there is no match with the listed values) or a VER statement (if the verification fails).

**.RESP** Indicates "normal" or "exception" response on the part of the user. It is always set to "ENTER" (normal response) unless the user enters an END or RETURN command, in which case it is set to "END". It may be tested in the processing section to determine the user's response. For 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 though the user had pressed ENTER or entered an END command without entering any data.

This variable may be set in a panel processing section to force an END or ENTER response. This may be particularly useful if a verification has failed (or .MSG set) and it is desired that the panel be redisplayed with the message even if the user entered END or RETURN.

**.TRAIL** Contains the "remainder" following a truncate (TRUNC) operation. If the contents of a variable are truncated to a specified length, all remaining characters are stored in .TRAIL. If the contents of a variable are truncated at the first occurrence of a special character, the remaining characters following the special character are stored in .TRAIL.

**.ZVARS** May be set in the initialization section to a list of variable names that correspond to "Z" placeholders in the body and/or model lines. See "Z Variables as Field Name Placeholders."

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 by the calling sequence (if any) for an initial message or cursor position. Under certain conditions, processing of the initialization section is bypassed. See "Processing Considerations."

Once .CURSOR, .MSG, and .RESP have been set non-blank, they retain their initial values until the panel is displayed (or redisplayed), at which time they are again reset.

Figure 29 shows an example in which both .HELP and .CURSOR have been set in the )INIT section of the panel definition.

*.ALARM = { YES }  
          { NO }*

*.PFKEY*

*.ATTR*

*.ATTRCHAR*

```

)BODY
%----- EMPLOYEE RECORDS -----
%COMMAND==> _ZCMD +
+
%EMPLOYEE SERIAL: &EMP SER
+
+ 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 29. 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 is 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.
  - It must not have an initial value (i.e., the corresponding dialog variable must be null or blank).
  - It must not have a field name of `ZCMD`.
- If no fields meet the above criteria, the cursor is placed on the first input field in the panel body (normally the command field).
- If the panel has no input fields, the cursor is placed at row 1, column 1.

Whenever a message is displayed because of a verification failure, a `MSG=value` condition in a `TRANS`, or an explicit setting of `.MSG`, the cursor is automatically positioned at the beginning of the field that was last referenced in any panel definition statement. For example:

```
&XYZ = TRANS (&A ... MSG=xxxxxx)
&A = TRANS (&XYZ ... MSG=xxxxxx)
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 the above examples, the cursor would be placed on field `XYZ` if a message is displayed.

## **"Z" Variables as Field Name Placeholders**

In the body section of a panel definition and in the model lines for a table display panel, the name of an input or output field may be represented by the single character `"Z"`. This serves as a placeholder; the actual name of the field is defined in the initialization section of the panel definition.

Use of placeholders allows the definition of short fields for which the lengths of the variable names exceed the lengths of the fields.

The actual names of these fields are assigned in the initialization section of the panel definition using a name list (enclosed in parentheses if more than one name is specified) assigned to the control variable `.ZVARS`. The first name in the list corresponds to the first `"Z"` placeholder that appears in the body or model lines. The second

name in the list corresponds to the second "Z", etc. An example is shown in Figure 30.

---

```
)BODY
----- TITLE LINE -----
%COMMAND====>_ZCMD
.
.
.
+ TYPE %====>_Z+ (A for alpha, N for numeric)
+ LENGTH%====>_Z + (0 to 99)
+ OFFSET%====>_Z + (0 to 99)
.
.
)INIT
.ZVARS = '(TYPFLD LNGFLD OFFSET)'
```

Figure 30. Example of "Z" Variable Placeholders

---

In this example, the input field labeled "type" is one character long and the next two input fields are each two characters long. The names of these three fields are TYPFLD, LNGFLD, and OFFSET, respectively.

The name list assigned to .ZVARS must be enclosed in apostrophes because the list contains special characters (parentheses) and blanks. As with other name lists, either commas or blanks may be used to separate the names in the list. The length of the entire list, including the parentheses, is limited to 255 characters.

### Panel Processing Considerations

When the DISPLAY service is invoked from a dialog function, the panel name, message-id, and cursor field parameters may be specified.

If the panel name or message-id is specified, 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 situations, 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 overrides an initial message or cursor position passed by the calling sequence parameters.

In the third and fourth situations, processing of the )INIT section is bypassed, and there is no automatic initialization of variables in the panel body (nor in the attribute section). As a result, all variables in the panel body appear as last displayed, and input fields contain whatever the user last entered. If an initial message or cursor position is passed in the calling sequence parameters, that setting is 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 enter an End or RETURN command. In this case, 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.

## Special Panel Requirements

Special requirements exist for defining the following types of panel:

- Menus
- Help tutorials
- Table displays

## Menus

A menu (also called a selection panel) is a special type of panel. It is processed by the SELECT service. A menu must have an input field to be used or entry of selection options by the user of the application. Generally, this field (called the command field) is the first input field on line 2 of the panel. Name this field ZCMD to be consistent with the name used in this publication. (OPT is the name of this field used in SPF, the ISPF predecessor product). It must also have a processing section in which variable ZCMD is truncated at the first period and then translated to a character string. The results must be stored in a variable named ZSEL or SEL. (SEL is provided only for compatibility with SPF; use of ZSEL is recommended.)

Besides ZCMD, a menu may have input fields to set up dialog variables needed by the particular application. Any variables other than ZCMD and ZSEL (or OPT and SEL) that are set from a menu are automatically stored in the shared variable pool.

Variables from the shared pool (including system variables) may also be displayed on a menu to provide information to the end user.

The general format of the processing section of a menu is:

```
)PROC
  &ZSEL = TRANS( TRUNC(&ZCMD, '.')
                value, 'string'
                value, 'string'
                . . .
                value, 'string'
                ' ', ' '
                *, '?' )
```

Note - Allows blank input to be ignored

*anything else"*

The ZCMD variable is truncated prior to translation to allow the end user to bypass one or more intermediate menus. For example, "1.2" means primary option 1, suboption 2. (This is generally called a "nested option.") When the SELECT service discovers that variable ZCMD (which was automatically stored, untranslated, as the user entered it) contains a period, it causes 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 is processed in the normal fashion but not displayed to the end user.

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 are:

blank is legal (although not a "string")

```
'PANEL(panel-name) [NEWAPPL(application-id)|NEWPOOL]'  
'CMD(command) [NEWAPPL(application-id)|NEWPOOL] [NOCHECK]'  
'PGM(program-name) [PARM(parameters)]  
[LANG(PLI|PL1 [,storage-area])]  
[LANG(COBOL)]  
[NEWAPPL(application-id)|NEWPOOL] [NOCHECK]'  
EXIT
```

Except for EXIT, each string of keywords must be enclosed in apostrophes, since it contains parentheses (and sometimes blanks).

The following selection keywords are the same as those that may be specified for the SELECT service (for a description of these keywords see "SELECT" in Chapter 6):

```
PANEL(panel-name)  
CMD(command)  
PGM(program-name)  
PARM(parameters)  
[LANG(PLI|PL1 [,storage-area])]  
[LANG(COBOL)]  
NEWAPPL(application-id)|NEWPOOL
```

→ p. 140

The PANEL keyword, for example, is used to specify the name of a lower-level menu to be displayed. The CMD and PGM keywords are used to invoke a dialog function coded in a command procedure or programming language, respectively. Note that OPT(option), which is valid for the SELECT service, is not valid on the definition of a menu. NOCHECK and EXIT are described below.

Normally, nested options are allowed only when each component of the option (up to, but not including the last component) specifies a lower-level menu. For example, given the following ZSEL keywords on a selection panel definition

```
&ZSEL = TRANS (TRUNC(&ZCMD, '.'))  
1, 'PANEL(DEF)'  
.....  
8, 'PGM(ABC)'  
9, 'PGM(XYZ)'
```

A user may enter "1.x" as his selection. This selection would be accepted by ISPF. However, if the developer wants to allow a user to enter a nested option that selects a dialog function (in this case "8.x" or "9.x"), the developer specifies the NOCHECK keyword as in the following example:

```

&ZSEL = TRANS (TRUNC(&ZCMD, '.'))
           1, 'PANEL(DEF)'
           .....
           8, 'PGM(ABC) NOCHECK'
           9, 'PGM(XYZ) NOCHECK'

```

The NOCHECK keyword specifies that normal checking for validity is suspended. It is the responsibility of the dialog function to interpret the meaning of the lower-level options. To allow this, the remaining options (those to the right of the first period) are normally passed to the dialog function through any appropriate variable that has been set equal to the .TRAIL panel control variable in the menu definition. Example:

```

&ZSEL = TRANS (TRUNC (&ZCMD, '.'))
           1, 'PANEL(DEF)'
           8, 'PGM(ABC) NOCHECK'
           9, 'PGM(XYZ) NOCHECK'
&NEXTOPT = .TRAIL

```

In this example, variable NEXTOPT contains the remainder of the TRUNC operation. If the user enters "8.5.2", program ABC is invoked and NEXTOPT is set to "5.2". If the user enters "9.7", program XYZ is invoked and NEXTOPT is set to "7". Since variable NEXTOPT is unknown to the SELECT service, it is automatically stored in the shared variable pool, where it can be accessed by the dialog function.

When the NOCHECK keyword is specified, a return code of 20 from the dialog function indicates that the remaining options are invalid. If return code 20 is passed back from the function, an "invalid option" message is displayed by the dialog manager.

The EXIT keyword, if used, applies only to a primary option menu. It terminates ISPF using defaults for list/log file processing.

If no option is entered (ZCMD variable is blank), a blank is returned as the translated string. This causes the SELECT service to redisplay the menu. For primary option menus, the menu is redisplayed without a message. For lower-level menus, an "enter option" message is displayed if the option field was left blank.

If an invalid option is entered (indicated by an asterisk, meaning none of the above), a question mark (?) is returned as the translated string. This causes the SELECT service to redisplay the menu with an "invalid option" message.

## Primary Option Menus

A primary option menu is a selection panel that has special significance in terms of the way the RETURN command operates, and in terms of the way a "jump function" (an option number preceded by an equal sign) works. One type of primary option menu is the master application menu. It is described in "Examples of Menus" below.

The first menu displayed when the dialog manager is invoked is normally treated as a primary option menu. That is, if the dialog manager is invoked with:

```
ISPSTART PANEL(XYZTOP)
```

then panel XYZTOP is treated as a primary option menu.

Similarly, if the dialog manager is invoked with:

```
ISPSTART CMD(XYZ) or  
ISPSTART PGM(XYZ)
```

and dialog XYZ subsequently issues:

```
SELECT PANEL(XYZTOP)
```

then panel XYZTOP is treated as a primary option menu because it is the first invoked menu.

It is possible to write a dialog with no primary option menu by setting the variable ZPRIM to "NO" on the first selection panel (panel XYZTOP in this example):

```
)INIT  
&ZPRIM = NO
```

In general, this is not recommended since the RETURN command then causes an immediate exit from the dialog, which may be confusing to the end user.

A dialog may have lower-level (nested) primary option menus. This is achieved by setting variable ZPRIM to "YES" on a lower-level selection panel:

```
)INIT  
&ZPRIM = YES
```

Nested primary option menus should be used sparingly, since they can confuse the end user. It is recommended that there be only one primary option menu per major application.

## Set Next Menu

ISPF allows the display of menus that are arranged in a hierarchy. The path through the hierarchy is automatically preserved as the user selects options from the various menus. As the user moves back up through the hierarchy, the menus are redisplayed in reverse sequence from which they were selected. While this is the standard mode of operation, it may be overridden. A developer may specify an alternative mode of menu processing called explicit chain mode. In this mode, the "parent" menu is specified explicitly in a system variable named ZPARENT. This variable may be set in a panel definition or in a dialog function. It has the following effect:

- From a menu, ZPARENT specifies the name of the next menu to be displayed when the user enters the END command. A menu that specifies itself as the parent is interpreted as a primary option menu; the RETURN command "stops" at that menu.
- From a function, ZPARENT specifies the name of the next menu to be displayed when the function completes execution. If a function is invoked from another function (by the SELECT service), the lower-level function may set ZPARENT. Upon completion of the lower-level function, the higher-level function resumes execution. The setting of ZPARENT does not take effect until the higher-level function (i.e., the one originally invoked from a menu) completes execution.

### Notes:

1. A value may be stored in ZPARENT in a function or in the )INIT, )PROC, or )BODY section of a panel.
2. The value in ZPARENT takes effect only after display of a selection panel when the user enters the END command.
3. When the ZPARENT variable is set from a dialog function, it must be explicitly copied to the shared pool (using VPUT) to ensure that ISPF still has access to it after the function completes.
4. Once the ZPARENT variable is set:
  - The hierarchy of menus traversed by the user is not preserved by ISPF.
  - The NEWAPPL and NEWPOOL selection keywords are inoperable (ignored) while the dialog is in explicit chain mode.
5. The ZPARENT variable is automatically reset to blank by the dialog manager each time it is used. If the dialog does not continue to set ZPARENT, the dialog manager resumes normal mode. The hierarchy of menus (if any) up to the point at which explicit chain mode was started is then restored.

6. Generally, a dialog should use either explicit chain mode or hierarchical chaining (the standard mode); chaining should not be mixed. If they are mixed, a function that sets ZPARENT should do so only after completion of any hierarchical dialog that it invokes (i.e., the setting of ZPARENT should be the last thing the function does before returning control). Otherwise, results are unpredictable.
7. The ZPRIM variable is not applicable to (and is ignored) when operating in explicit chain mode.

### Examples of Menus

Typical menus are described below.

MASTER APPLICATION MENU: A master application menu, shown in Figure 31, and named ISP@MSTR, is distributed with ISPF as part of the panel library. This menu may be used, if desired, to allow selection of the various applications available at an installation.

```

%----- ISPF MASTER APPLICATION MENU -----
%OPTION ==> _ZCMD +
%
%
%                               +USERID   - &ZUSER
%    1 +SAMPLE1          - Sample application 1           +TIME     - &ZTIME
%    2 +.                - (Description for option 2)    +TERMINAL - &ZTERM
%    3 +.                - (Description for option 3)    +PF KEYS  - &ZKEYS
%    4 +.                - (Description for option 4)
%    5 +.                - (Description for option 5)
%    X +EXIT             - Terminate ISPF using list/log defaults
%
+Enter%END+command to terminate ISPF.
%
)INIT
  .HELP    = ISP00005       /* Help for this master menu      */
  &ZPRIM = YES              /* This is a primary option menu  */
)PROC
  &ZSEL = TRANS( TRUNC (&ZCMD, '.')
                1, 'PANEL(ISP@PRIM)' /* Sample primary option menu */
                /*****/
                /*                                     */
                /* Add other applications here.         */
                /*                                     */
                /*****/
                /* Following shows how to code an invocation of the   */
                /* ISPF Program Development Facility, where "n" is     */
                /* the desired selection number:                   */
                /*                                     */
                /* n, 'PANEL(ISR@PRIM) NEWAPPL(ISR)'          */
                /*                                     */
                /*****/
                ' ' ' '
                 1 1 1 1
                X, 'EXIT'
                 *, '? ' )
)END

```

Figure 31. Master Application Menu

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:

```
ISPSTART [PANEL(ISP@MSTR)]
```

When no keywords are specified on the ISPSTART command, PANEL (ISP@MSTR) is assumed. 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.



PRIMARY OPTION MENU: Figure 32 shows a primary option menu definition. This is the sample primary option menu definition (ISP@PRIM) distributed with ISPF. The required input field, ZCMD, appears in the second line of the panel body. It is followed by a description of the various options available to the user.

This menu also has four variables within text fields at the upper right-hand 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 enters the HELP command from this menu. It also initializes two system variables that specify the tutorial table of contents and first index page. See the discussion under "Help/Tutorial Panels."

**Note:** &ZPRIM=YES specifies that this panel is a primary option menu.

The processing section specifies the action to be taken for each option entered by the user. If option 0 is selected, panel ISPOPTA (a lower level menu) is displayed. If option 1 is selected, panel ISPUCMA is displayed; and so on.

Note that for the tutorial, program ISPTUTOR is invoked and passed a parameter (ISPO0000), which ISPTUTOR interprets as the name of the first panel to be displayed. Panel ISPO0000 is the first panel in the tutorial for ISPF. Other applications should pass the name of the first tutorial page for that application.

```

%----- SAMPLE PRIMARY OPTION MENU -----
%OPTION ==> _ZCMD
%
%
%          +USERID   - &ZUSER
% 0 +ISPF PARMS - Specify terminal and user parameters +TIME       - &ZTIME
% 1 +COMMANDS   - Create/change command table         +TERMINAL - &ZTERM
% 2 +.          - (Description for option 2)          +PF KEYS  - &ZKEYS
% 3 +.          - (Description for option 3)
% 4 +.          - (Description for option 4)
% 5 +.          - (Description for option 5)
% T +TUTORIAL   - Display information about this application
% X +EXIT       - Terminate ISPF using list/log defaults
%
+Enter%END+command to terminate application.
%
)INIT
  .HELP = ISPO0003      /* Help for this panel          */
  &ZPRIM = YES          /* This is a primary option menu */
  &ZHINDEX = ISP91000  /* Tutorial index - 1st page for this appl */
  &ZHTOP = ISPO0003    /* Tutorial table of contents for this appl */
)PROC
  &ZSEL = TRANS( TRUNC (&ZCMD, '.')
                0, 'PANEL(ISPOPTA)'
                1, 'PANEL(ISPUCMA)'
                /*****
                /*
                /* Add other applications here.
                /*
                *****/
                T, 'PGM(ISPTUTOR) PARM(ISPO0000)'
                , ' '
                , ' '
                X, 'EXIT'
                *, '?' )
)END

```

*licable to P-processing*

Figure 32. ISPF Primary Option Menu

**LOWER-LEVEL MENUS:** Lower-level menus follow the same rules as a master or primary option menu. An example of a lower-level menu is shown in Figure 33. This menu is used in the MVS version of ISPF/PDF. The panel is ISRUTIL, which is displayed if option 3 is selected from the ISPF/PDF primary option menu. For option 1, it specifies that program ISRUDA is to receive control, and that ISRUDA is to be passed a parameter (UDA1) which ISRUDA 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 ZCMD need not have been truncated prior to translation, since there is no lower-level selection panel that can be displayed from this menu.

```

%----- UTILITY SELECTION MENU -----
%OPTION ==>_ZCMD
%
% 1 +LIBRARY - Library utility:
+          Print index listing or entire data set
+          Print, rename, delete, or browse members
+          Compress data set
% 2 +DATASET - Data set utility:
+          Display data set information
+          Allocate, rename, or delete entire data set
+          Catalog or uncatalog data set
% 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 ISPF 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 +COMMANDS - Create/change an application command table
% 10 +CONVERT - Convert old format menus/messages to new format
)INIT
  .HELP = ISR30000
)PROC
  &ZSEL = TRANS( TRUNC (&ZCMD, '.')
    1, 'PGM(ISRUDA) PARM(UDA1)'
    2, 'PGM(ISRUDA) PARM(UDA2)'
    3, 'PGM(ISRUMC)'
    4, 'PGM(ISRUC)'
    5, 'PGM(ISRURS)'
    6, 'PGM(ISRUHC)'
    7, 'PGM(ISRUVT)'
    8, 'PGM(ISRUOL)'
    9, 'PANEL(ISPUCMA)'
    10, 'PGM(ISRQCM) PARM(ISRQCM)'
    *, '?' )
)END

```

Figure 33. Lower-Level Menu

## Help/Tutorial Panels

A help or tutorial panel is a special type of panel that is processed by the ISPF tutorial program, which invokes the panel display service to display the panel. The tutorial program may be invoked either from a menu, or through the HELP command.

Tutorial panels are arranged in a hierarchy. Generally, this hierarchy is a table of contents, each succeeding level of which contains a succeeding more detailed list of topics. When the tutorial is entered from a 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.

When the tutorial is entered by use of the HELP command, the first panel to be displayed is a panel within the hierarchy, appropriate to what the user was doing when help was requested. In this situation, 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 pressing the ENTER key to view the next topic. On any panel, the user may also enter the following commands:

BACK or B - to return to the previously viewed panel

SKIP or S - to advance 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, as shown in Figure 32.

The index is optional and is a collection of panels in which topics are arranged in alphabetic order. A user may jump to the index from any point by the use of the INDEX command. The index need not be connected to the main tutorial hierarchy; it may be a selectable topic from the main table of contents or other panels.

ZHTOP - P. 237

Each tutorial panel must have a "next selection" input field. Generally, you should use the name ZCMD for this field. A tutorial panel should also have a processing section in which the following variables are set:

**ZSEL or SEL** Specifies the name of the next panel to be displayed based on the topic selected by the user (by translating ZCMD 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 is bypassed if the user presses the ENTER key to view the next topic.

If a panel does not have any selectable topics, ZSEL should be omitted.

**ZUP or UP** Specifies the name of the parent panel, from which this panel was selected. Generally, ZUP may be omitted since the tutorial program remembers the sequence of selections that lead to the display of this panel. ZUP is used only if this panel is the first to be displayed (by a user entering the HELP command) or is selected from the tutorial index, and the user then enters the UP command.

**ZCONT or CONT** Specifies the name of the next continuation panel. If there is no continuation panel, ZCONT should be omitted.

**ZIND** When set to a value of YES, specifies that a page in the tutorial is an index page. For example:

```
)PROC
  &ZIND = YES
```

The ZIND variable is used only on index pages; it should not be set on other tutorial panels.

**Note:** Variables SEL, UP, and CONT are provided for compatibility with the previous SPF product. Use of variable names ZSEL, ZUP, and ZCONT is recommended. Refer to Appendix H, "MVS and VM/SP: Summary of Changes From SPF."

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.

Help/Tutorial panels may contain variables so that dialog information (including information entered by the user) may be displayed on the help panel. Function variables, as well as shared and profile variables, may be displayed.

Figure 34 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).

In Figure 34, assuming that panel A is the highest-level table of contents, the viewer can get to A from any point by issuing TOP. A viewer currently on panel F1, F2, or F3 may return to panel B by issuing BACK. Then, from B, the SKIP command would take the viewer to panel C.

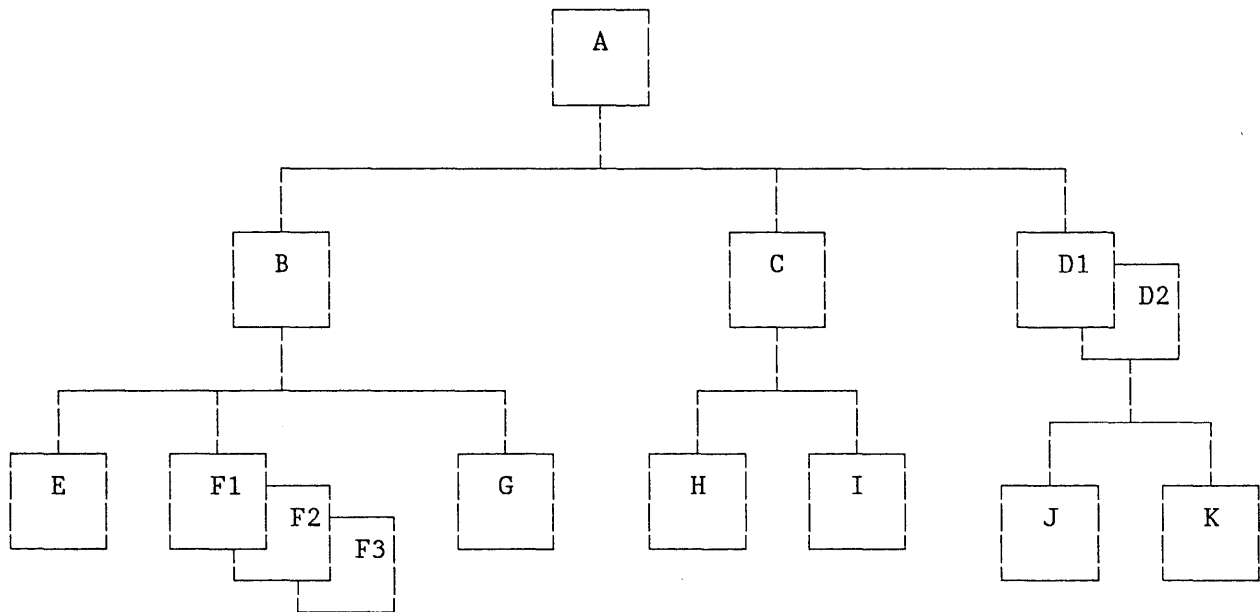


Figure 34. Sample Tutorial Hierarchy

Two sample tutorial panels are shown in Figure 35 and Figure 36. These are assumed to be panels B and F2, respectively, in the hierarchy in Figure 34.

Panel B has three selectable topics. In the processing section, ZCMD is translated to a panel name (E, F1, or G) corresponding to the selected option, and the result is stored in ZSEL. If none of the valid options is selected, a question mark (?) is returned as the translated string, which causes the tutorial program to display an "invalid option" message.

Note that option 3 is translated to "\*G". This indicates that panel G is displayed if the user selects option 3, but is 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 ZUP.

Panel F2 (Figure 36) has no selectable topics, but does have a continuation page. The name of the continuation panel (F3) is stored in variable ZCONT. The name of the parent panel (B) could have been stored in ZUP, but this was omitted assuming that F2 cannot be directly entered by use of the HELP command or from the tutorial index.

---

```
%TUTORIAL ----- 3270 DISPLAY TERMINAL -----TUTORIAL
%NEXT SELECTION ==> _ZCMD +
%
```

```
-----
|           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
```

```
&ZSEL = TRANS(&ZCMD 1,E 2,F1 3,*G *,'?')
```

```
&ZUP = A
```

```
)END
```

Figure 35. Sample Tutorial Panel (B)

---

---

```

%TUTORIAL ----- ERASE EOF KEY ----- TUTORIAL
%NEXT SELECTION ==> _ZCMD
+
+
When the erase EOF (erase to end-of-field) key is used, it will appear
to blank out the field.  Actually, 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 an input
field, the keyboard will lock.  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
  &ZCONT = F3
)END

```

Figure 36. Sample Tutorial Panel (F2)

---

### Table Display Panels

A table display panel is a special panel that is processed by the TBDISPL service. The panel definition contains non-scrollable text, including column headings, followed by one or more model lines. These lines describe how each table row is to be formatted within the scrollable data area. Attribute characters in the model lines indicate whether each field is protected or user-modifiable.

If a single model line is specified in the panel definition, each row from the table is mapped to the format of that line. This results in scrollable data that is in tabular format. For many applications, it may be useful to define the left-most column in each line as an input field where the application user may enter a code to be used by the dialog function to determine the particular processing for that row.

If multiple model lines are specified in the panel definition, each row from the table is mapped to multiple lines on the screen. (If desired, a separator line - consisting of blanks or dashes, for example - may be specified as the first or last model line.) This format may be useful for address lists or other repetitive data in which each unit will not fit on a single line.



PANEL DEFINITION REQUIREMENTS: 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 lines. For the model lines, only the attributes TYPE, INTENS, and PAD are meaningful; all fields in the model line assume CAPS(OFF) and JUST(ASIS).: An attribute section is required if the model line 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 eight characters long. The command field may have any desired name.

The command field is used (the same as on other types of panels) to enter ISPF commands and application-defined commands (if any). Any commands entered in this field that are not recognized by ISPF 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 four 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 section. This section begins with a )MODEL header statement and is immediately followed by one or more model lines.

The )MODEL header statement must begin in column 1. The following optional keywords may be specified on this header:

- CLEAR(var-name,var-name ...)
- ROWS(ALL|SCAN)

The CLEAR keyword identifies the dialog variable names, from the model lines, that are to be cleared (initialized to blank) before each row in the table is read. Thus, if the variable is an "extension" variable in the table, which may not exist in all rows, previous values are erased and are thereby not repeated in other lines for which they do not apply.

The ROWS keyword indicates whether all rows from the table are to be displayed, or whether the table is to be scanned for selected rows to be displayed. The default is ROWS(ALL) which causes all rows to be displayed. If ROWS(SCAN) is specified, the dialog must invoke the TBSARG service prior to invoking TBDISPL. The search argument set up by

the TBSARG service is used to scan the table; only rows that match the search argument are displayed.

One or more model lines must appear following the )MODEL header statement. A model line may contain input and output fields. Each field consists of an attribute character followed by a variable name. This name may be the name of a variable in the table row, or may be the null system variable, "Z", which serves as a placeholder. "Z" variable name replacement is resolved within the )INIT section, as described below.

Typically, the first field within the model lines specifies the dialog variable into which a selection code (entered by the user) will be stored, and all remaining names correspond to columns in the table. However, this arrangement is not required; any name may or may not correspond to a column in the table and a selection code field need not be specified.

Text fields may be specified 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 are repeated 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 lines; the results are unpredictable.

A maximum of eight model lines is allowed.

Initialization Section (may be required): If "Z" variables occur as name placeholders within the model lines, an )INIT section is needed. The real names of these fields are defined by assigning a name list (enclosed in parentheses, if more than one name is given) to the control variable, .ZVARS. (See "Control Variables" for a description of the use of .ZVARS.) For example:

```
)INIT
.ZVARS = '(NAME1,NAME2,NAME3)'
```

where NAME1, NAME2, and NAME3 are the actual variable names corresponding to the first, second, and third "Z" variables in the model lines.

**Note:** For compatibility with SPF, "Z" variables in the model lines of a table display panel may be assigned to the VARS variable, rather than to the control variable, .ZVARS. For example:

```
)INIT
  &VARS = '(NAME1,NAME2,NAME3)'
```

It is recommended, however, that existing table display panels be converted to use the new .ZVARS control variable. They must be converted if the new CLEAR keyword is added to the )MODEL header statement or explicit cursor positioning is used for table display.

The initialization section may contain any statement that is valid in an )INIT section of a panel definition. However, only the .CURSOR, .HELP, and .ZVARS control variables may be set.

Processing Section (omit): A table display panel should not contain a processing section; if it does, the results are unpredictable.

**TBDISPL PROCESSING:** When a panel is displayed by the TBDISPL service, the model lines in the )MODEL section are duplicated to the end of the logical screen. When the scrollable portion of the screen is being formatted, only full units or duplications of these model lines are displayed. 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 to by the CRP when TBDISPL was issued.

Input or output fields in a 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 invocation of TBDISPL, or the CLEAR keyword may be used to specify that they are to be blanked.

The user may scroll the data up and down. Scroll commands (e.g., "DOWN 5") apply to the number of table entries to scroll up or down. Example: If three model lines are specified, "DOWN 5" would scroll by 5 table entries, which corresponds to 15 lines on the display.

The user may enter information in the command field and in any of the input fields within the rows. Processing of input is described in the TBDISPL service description in Chapter 6, "Description of Services."

Figure 37 shows a sample panel definition for table display. Assuming that the current contents of the table are as shown in Figure 38, the resulting display is shown in Figure 39.

```

)ATTR
  @ TYPE(OUTPUT) INTENS(LOW)
)BODY
%----- EMPLOYEE LIST -----
%COMMAND INPUT ==>_ZCMD                                %SCROLL ==>_AMT +
+
+SELECT      ----- EMPLOYEE NAME -----          -- PHONE ---      EMPLOYEE
+ CODE      LAST          FIRST          MI          AREA NUMBER      SERIAL
)MODEL
  _Z+      @LNAME          @FNAME          @I          @PHA @PHNUM          @EMP SER
)INIT
  .ZVARS = '(SELECT)'
  &AMT = PAGE
  .HELP = PERS123
)END

```

Figure 37. Table Display Panel Definition

---

| EMP SER | 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 38. 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 39. Table as Displayed

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 model line 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.

The same table might be displayed using multiple model lines with the panel definition shown in Figure 40. The resulting display is shown in Figure 41. An entry separator, consisting of a dashed line, is also included as the last model line. In this example, the SELECT field has been increased to 4 characters, with underscores used as pad characters.

---

```

)ATTR
@ TYPE(OUTPUT) INTENS(LOW)
# TYPE(INPUT) PAD('_')
)BODY
%----- EMPLOYEE LIST -----
%COMMAND INPUT ==> _ZCMD                                %SCROLL ==> _AMT +
+
+ENTER CHANGES ON THE LINES BELOW.
+
)MODEL
#Z  + SERIAL: @EMPSER +                                LAST NAME: @LNAME      +
      PHONE: @PHA@PHNUM  +                            FIRST NAME: @FNAME     +
  INITIAL: @I+
-----
)INIT
.ZVARS = '(SELECT)'
&AMT = PAGE
.HELP = PERS123
)END

```

Figure 40. Table Display Panel Definition with Multiple Model Lines

---

---

----- EMPLOYEE LIST ----- LINE 000001 COL 001 080  
COMMAND INPUT ==> \_ SCROLL ==> PAGE

ENTER CHANGES ON THE LINES BELOW.

\_\_\_\_ SERIAL: 598304 LAST NAME: Robertson  
PHONE: 301 840-1224 FIRST NAME: Richard  
INITIAL: P

-----  
\_\_\_\_ SERIAL: 172397 LAST NAME: Smith  
PHONE: 301 547-8465 FIRST NAME: Susan  
INITIAL: A

-----  
\_\_\_\_ SERIAL: 813058 LAST NAME: Russell  
PHONE: 202 338-9557 FIRST NAME: Charles  
INITIAL: L

-----  
\_\_\_\_ SERIAL: 395733 LAST NAME: Adams  
PHONE: 202 477-1776 FIRST NAME: John  
INITIAL: Q

-----  
\_\_\_\_ SERIAL: 502774 LAST NAME: Caruso  
PHONE: 914 294-1168 FIRST NAME: Vincent  
INITIAL: J

-----  
\*\*\*\*\* END OF DATA \*\*\*\*\*

Figure 41. Table as Displayed with Multiple Model Lines

---

## MESSAGE DEFINITIONS

ISPF message definitions are stored in a message library and displayed by means of the DISPLAY, TBDISPL, or SETMSG service, or written to the ISPF log file by the LOG service. Messages are created or changed by editing directly into the message library. The messages are interpreted during ISPF processing; no compile or preprocessing step is required.

### Message ID

Each message is referenced by message-id. A message id may be four to eight characters long, and is defined as follows:

- Prefix: one to five alphabetic characters (A-Z, #, \$, or @)
- Number: three numeric characters (0-9)
- Suffix (optional): one alphabetic character

If the prefix is five characters long, the suffix must be omitted so that the total length does not exceed eight characters.

### Message Library

Several messages may be contained within each member of the message library. When using ISPF/PDF EDIT to create a message file, NUMBER OFF should be specified.

The member name is determined by truncating the message-id after the second digit of the number. For example:

| <u>Message id</u> | <u>Member name</u> |
|-------------------|--------------------|
| G015              | G01                |
| ISPE241           | ISPE24             |
| XYZ123A           | XYZ12              |
| ABCDE965          | ABCDE96            |

All messages that have ids beginning with the characters "G01", for example, must be in member G01. Figure 42 shows an example of a member in the message library. This member contains all message-ids that begin with "EMPX21".

Within the member, messages generally should appear in collating sequence by message-id. The optional message id suffix should be used if more than 10 messages are to be included in one member.

Each message in the library consists of two lines, as follows:

```
msgid ['short message'] [.HELP = panel|*] [.ALARM = YES|NO]
'long message'
```



---

```
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 '&EMPSEER ADDED'
'EMPLOYEE &LNAME, &FNAME &I ADDED TO FILE.'
```

```
EMPX218 '&EMPSEER UPDATED'
'RECORDS FOR &LNAME, &FNAME &I UPDATED.'
```

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

Figure 42. Sample Member in Message Library

---

The short message is optional. If a short message is specified on an ISPF panel, it is displayed first. Short messages are right-justified and displayed at the right end of the first line on the screen. If the user enters the HELP command, the long message is displayed on the third line of the screen. If the user enters the HELP command again, tutorial mode is entered.

The long message is required. If a short message is not specified, the long message is displayed first, on the third line of the screen. If the user then enters the HELP command, tutorial mode is entered.

The location of the short and long messages in a user-designed panel is specified by the SMSG and LMSG keywords. These keywords are defined in "Panel Body Section."

If tutorial mode is entered by the user, the panel name specified by .HELP is the first tutorial page displayed. If .HELP=\* is specified, the first tutorial page is whatever was specified in the panel definition (i.e., the panel on which this message is being displayed). The default is "\*".

If .ALARM=YES is specified, the audible alarm is sounded whenever the message is displayed. If .ALARM=NO is specified, the alarm is not sounded. The default is NO.

When messages are written to the ISPF 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. For 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.

Substitutable parameters in messages are replaced with values immediately before display or, if the message is specified for display using the SETMSG service, substitutable parameters are replaced during SETMSG processing. After substitution of the variables, the short message is truncated to 24 characters and the long message is truncated to 78 characters.

## Syntax Rules

The following rules apply to the syntax of messages as they appear in the message library.

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; for example:

```
'ENTER &X, &Y, OR &Z'
```

where a comma terminates the variable names X and Y. The name Z is delimited by the apostrophe that marks the end of the message.

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; for example:

&& yields &  
' ' yields ' within delimiting apostrophes  
.. yields . immediately following a variable name.

## SKELETON DEFINITIONS

ISPF skeleton definitions are stored in a skeleton library and accessed by means of the ISPF file tailoring services. Skeletons are created or changed by editing directly into the skeleton library. The skeletons are interpreted during ISPF execution; no compile or preprocessing step is required.

**Note:** The ISPF-distributed skeleton library also contains old format Structured Programming Facility "proc" members. (The Structured Programming Facility is a predecessor program product to SPF, which, in turn, is the predecessor of ISPF.)

The following description of skeleton formats applies only to new format skeletons used with ISPF file tailoring services.

There are two types of records that may appear in the skeleton file:

1. Data Records - a continuous stream of intermixed text, variables, and control characters that are processed to create an output record.
2. Control Statements - to 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. If the result of variable substitution is an output record larger than 80 characters, file tailoring is terminated and a message is displayed.

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 situation, 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.

Any blank data records are deleted from file tailoring output.

The following control characters have special meanings:

- An exclamation point (!) 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 point in the input record is put at the tab stop location in the output record. Tab stops are specified by use of 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.

## Control Statements

The general format of a control statement, which must begin in column 1, is:

```
)control-word token1 token2 ... 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, preceded by an ampersand
- 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 "Data Records," 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 effect immediately, when it is encountered. It remains in effect until the end of FTINCL processing, or until another )DEFAULT statement is encountered.

```
)TB value1 ... value8
```

Up to eight 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 three 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 not be present. If OPT is coded and the skeleton is not present, no error

indication is given, and the record is ignored. If OPT is not coded, and the skeleton is not present, a severe error occurs.

)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 eight 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 eight 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
)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 four 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.

For an example showing use of )DOT and )ENDDOT, see "File Tailoring" in Chapter 2.

)SET variable = expression

)SET allows a value to be assigned to a dialog variable. The variable name should not 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.

**Note:** The )N comment statement of PDF edit models is not a valid control statement for file tailoring. It causes file tailoring termination.

## Sample Skeleton File

A sample skeleton file is shown in Figure 43.

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 43. Sample Skeleton File

---





## APPENDIX A. USING THE DISPLAY SERVICE

This appendix describes the use of the DISPLAY, TBGET, and TBADD services in a dialog function that allows a user to add data to a table.

Sections of this publication, containing information needed to understand this appendix, are listed in the Reading List on the next page.

During function processing, the DISPLAY service is used to control displays requesting the user to enter data for new employees. The data consists of employee serial number (which is entered on panel SER) and name and phone number (which are entered on panel DATA). After the data is entered by the application user, it is added to the table, as a row, through use of the TBADD service.

If the application user enters an employee serial number for which an employee record already exists in the table, the message DUPLICATE NUMBER is displayed on line one of the panel, SER. If the user enters the HELP command or presses the HELP PF key 1, the message EMPLOYEE RECORD ALREADY EXISTS FOR THIS NUMBER. ENTER ANOTHER is displayed on line three of the panel.

When the user successfully enters data for an employee, the message NEW RECORD INSERTED is displayed on line one of panel SER. Then the user may enter the serial number of the next employee for which data is to be added to the table.

The user ends function processing by entering the END or RETURN command (or by pressing the END or RETURN PF key), on any panel that the function displays.

Logic that updates or deletes existing rows in the table is not included. Also, logic is not included to process any function commands originated by the user. (Logic that updates table rows and processes function commands is illustrated in Appendix C, "Using the TBDISPL Service" on page 289).

(The function may be initiated by a user from a terminal by means of the ISPSTART command. If the user has already started ISPF, the function may be initiated, at the terminal, from a menu (on which the function is to be invoked by one of the provided selections) or from any display containing a command line, by means of SELECT action in a command table. Or, the function may be initiated from another function by use of the SELECT service.)

Steps in dialog function processing are listed on a following page. Each step has a step identifier (1a, 1b, 2a, etc.). This identifier refers to a description, on a following page, of the processing performed by the step.

## READING LIST

Selected topics in this publication are listed below. They contain descriptions of the ISPF facilities illustrated by this appendix. The purpose is to allow a reader to gain, with a minimum of study, information to understand the example given on the following page.

### Sections to be Read

Chapter 1, in its entirety (7 pages)

Chapter 2, selected topics (15 pages):

- Dialog Organization
- Dialog Services Overview
  - Display Services
    - Panel Definitions
    - Message Definitions
  - Table Services
    - Table Residency
    - Accessing Data
    - Example
  - Variable Services
    - Variable Access - Order of Search
    - Relationship of Function Pools to Dialog Functions
    - The Function Pool for Command Procedures
    - The Function Pool for Programs (first paragraph only)

Chapter 3, selected topic (1 page):

- END and RETURN Commands (first two paragraphs only)

Chapter 6, selected topics (12 pages):

- Invocation of Services
  - Command Invocation
- Return Codes from Services
  - DISPLAY - Display Panels and Messages
  - TBADD - Add a Row to a Table
  - TBCLOSE - Close and Save a Table
  - TBGET - Retrieve a Row from a Table
  - TBOPEN - Open a Table

Chapter 7, selected topics (10 pages)

- Panel Definitions
  - Panel Body Section
    - Sample Body Section
  - Initialization and Processing Sections
    - Statement Formats
      - IF Statement
      - VER Statement
- Message Definitions
  - Message ID
  - Message Library

## STEPS IN FUNCTION PROCESSING

This section lists the steps in the function. Dialog service requests (CONTROL, TBOPEN, DISPLAY, etc.) issued in the steps, are in the command procedure format.

**Note:** Program format dialog service requests are illustrated with each service description in Chapter 6. Additional examples of services requests in both the command procedure (CLIST, EXEC) and program (COBOL, FORTRAN, PL/I) format may be found in ISPF Dialog Management Services Examples.

Additional comments about function processing follow in "Description of Steps in Function Processing."

| <u>Step</u><br><u>Id</u> | <u>Function Processing</u>      | <u>Comment</u>                                                                          |
|--------------------------|---------------------------------|-----------------------------------------------------------------------------------------|
| 1a.                      | CONTROL ERRORS CANCEL           | Terminate if 12 or higher return code.                                                  |
| 1b.                      | TBOPEN TAB1 WRITE               | Open table TAB1.                                                                        |
| 2a.                      | DISPLAY PANEL(SER)              | Display panel SER requesting serial number.<br>- User enters employee serial number.    |
| 2b.                      | if return code = 0, go to 3a    | Go retrieve any existing employee record.                                               |
| 2c.                      | if return code = 8, go to 8a    | END or RETURN entered.                                                                  |
| 3a.                      | TBGET TAB1                      | Attempt retrieval of employee record.                                                   |
| 3b.                      | if return code = 0, go to 4a    | 0 = record exists.                                                                      |
| 3c.                      | if return code = 8, go to 5a    | 8 = no record exists.                                                                   |
| 4a.                      | DISPLAY PANEL(SER) MSG(EMPX210) | Display DUPLICATE NUMBER message.<br>- User enters another number.                      |
| 4b.                      | if return code = 0, go to 3a    | Go retrieve any existing employee record.                                               |
| 4c.                      | if return code = 8, go to 8a    | END or RETURN entered.                                                                  |
| 5a.                      | Set dialog variables to blanks  | Blank variables LNAME, FNAME, I, PA, PHNUM.                                             |
| 5b.                      | DISPLAY PANEL(DATA)             | Display panel DATA requests employee data.<br>- User enters employee name and phone no. |
| 5c.                      | if return code = 0, go to 6a    | Go write data to the table.                                                             |
| 5d.                      | if return code = 8, go to 8a    | END or RETURN entered.                                                                  |
| 6a.                      | TBADD TAB1                      | Write new record to the table.                                                          |
| 6b.                      | if return code = 0, go to 7a    |                                                                                         |
| 7a.                      | DISPLAY PANEL(SER) MSG(EMPX211) | Display NEW RECORD INSERTED message.<br>- User enters next number to be added.          |
| 7b.                      | if return code = 0, go to 3a    | Go retrieve any existing employee record.                                               |
| 7c.                      | if return code = 8, go to 8a    | END or RETURN entered.                                                                  |
| 8a.                      | TBCLOSE TAB1                    | Write the table to permanent storage.                                                   |
| 8b.                      | End the function                | End the function.                                                                       |

## DESCRIPTION OF STEPS IN FUNCTION PROCESSING

The description below is related, by step id, to "Steps in Function Processing" above. Referenced figures are collected at the end of the description.

| <u>Step Id</u> | <u>Description</u>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1a             | This ISPF service request specifies that the function is to be terminated for a return code of 12 or higher from an ISPF service request.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 1b             | Open the table: read table TAB1 (contents of which are shown in Figure 44) into virtual storage.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 2a             | <p>This DISPLAY operation uses panel definition SER (shown in Figure 45) to control the format and content of the display (shown in Figure 46). The display requests the user to enter a serial number for an employee. After the user enters the serial number (in the field labeled EMPLOYEE SERIAL NUMBER), the DISPLAY service verifies it after storing it in function pool variable EMPSER. The verification is specified in a VER statement in the )PROC section of the panel definition, as shown in Figure 45:</p> <pre>VER (&amp;EMPSER, NONBLANK, PICT, NNNNNN)</pre> <p>This statement specifies that EMPSER must be nonblank and must consist of six digits, each in the range of 0 - 9.</p> <p>When the input passes the verification, the DISPLAY service returns control to the function.</p> <p>If the input fails the verification, the panel is automatically displayed again, but with an appropriate ISPF-supplied message displayed, right justified, on line 1.</p> <p>For example, if the user fails to enter the required employee serial number, an ISPF-provided message, ENTER REQUIRED FIELD, is displayed (shown in Figure 47).</p> <p>After reentry by the user, the information is stored again in function pool variable EMPSER and, again, it is verified. The process is repeated until the verification tests are passed.</p> <p>(Another example of verification processing is given in the step 5b description).</p> |
| 2b             | If the return code is 0, the display operation is successfully completed. Go to step 3a to verify that no record exists for this employee number.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

2c If the return code is 8, the END or RETURN command was entered on the display by the user. Go to step 8a to end processing.

3a This TBGET uses the employee serial number, stored in EMPSER in step 2a or 7a, to attempt retrieval of an employee record from the TAB1 table. The table is a keyed table and was created by the TBCREATE service, in another dialog, by the following request:

```
TBCREATE TAB1 KEYS(EMPSER) NAMES(LNAME FNAME I PHA PHNUM)
```

3b If the return code is 0, the record is found, which means that a record already exists for the employee serial number entered by the user. Go to step 4a to display the DUPLICATE NUMBER message.

3c If the return code is 8, no record is found. Go to step 5a to request the user to enter employee data.

4a This DISPLAY operation uses panel definition SER (shown in Figure 45) and message EMPX210 (shown in Figure 48) to control the format and content of the display.

**Note:** The following DISPLAY request, omitting the PANEL(SER) parameter, could have been used in this step:

```
DISPLAY MSG(EMPX210)
```

When the PANEL parameter is omitted, the specified message is superimposed on the panel currently being displayed, which, in this case, is the panel SER.

The short form of the message EMPX210, DUPLICATE NUMBER, is superimposed - right justified, on line 1 of the panel display (shown in Figure 49). While viewing this message, the user may enter the HELP command by pressing PF key 1. This causes the long form of the message to appear, superimposed - left justified, on line 3 of the display, as follows: EMPLOYEE RECORD ALREADY EXISTS FOR THIS NUMBER. ENTER ANOTHER. (See Figure 50).

After the user enters the requested serial number, the DISPLAY service stores it in function pool variable EMPSER and verifies it as described for step 2a. After the input passes verification, the DISPLAY service returns control to the function.

4b If the return code is 0, the display operation is successfully completed. Go to step 3a to verify that no record already exists for this employee number.

4c If the return code is 8, the END or RETURN command was entered on the display by the user. Go to step 8a to end processing.

- 5a** These function pool variables are blanked to prepare to receive data for a new employee record.
- 5b** The DISPLAY operation uses panel definition DATA (shown in Figure 51) to control the format and content of the display (shown in Figure 52).
- The variables blanked in step 5a are displayed along with the new employee serial number, which was entered in step 2a or 7a. The user is asked to enter, in the blanked fields displayed on the screen, the name and phone number for the employee.
- After the user enters these fields, the DISPLAY service stores the input in function pool variables LNAME, FNAME, I, PHA, and PHNUM. Then, verification of the input is performed as specified in VER statements in the )PROC section of the panel definition (shown in Figure 51).
- The input fields may pass the verification tests. If they do, the DISPLAY service returns control to the function.
- The input fields may fail the verification tests. If they do, a short form message is displayed superimposed on line 1 of the display.
- The message may be provided by ISPF (as described for step 2a) or the number of the message displayed may have been specified in the VER statement that defined the verification test (see VER statements containing message-ids EMPX212, EMPX213, and EMPX214 in Figure 51). Where a message-id is specified, this message is displayed instead of an ISPF-provided message. In either case, if the user enters the HELP command, the long form of the message is displayed, left justified, on line 3.
- The text of the messages request reentry of information. When reentered, this information is stored again in function pool variables and, again, it is verified. The process is repeated until the verification tests are passed.
- 5c** If the return code is 0, the display operation is successfully completed. Go to step 6a to add the record to the table.
- 5d** If the return code is 8, the END or RETURN command was entered on the display by the user. Go to step 8a to end processing.
- 6a** This TBADD adds a row to table TAB1 by copying values from function pool variables to the table row. The values copied are employee serial number (stored in the function pool variable EMPSER by step 2a or 7a) and employee name and phone number (stored in function pool variables LNAME, FNAME, I, PHA, and PHNUM by step 5b).

**Note:** Function pool variables must have the same names as the table variables to which they are to be copied by the TBADD operation.

Therefore, the names used in the TBCREATE that establishes the table (see the names EMPSER, LNAME, FNAME, I, PHA, and PHNUM in the TBCREATE illustrated in the step 3a description) are the same as the names used in the panel definitions (shown in Figure 45 and Figure 51) that establish function pool variables in which user-entered information is stored by the DISPLAY service.

- 6b** If the return code is 0, the TBADD operation is successfully completed. Go to step 7a to display the NEW RECORD INSERTED message.
- 7a** This DISPLAY operation uses panel definition SER (shown in Figure 45) and message EMPX211 (shown in Figure 48) to control the format and content of the display. The short form of message EMPX211, NEW RECORD INSERTED, is superimposed - right justified, on line 1 of the display. If the user enters the HELP command while this message is being displayed, the long form of the message (also shown in Figure 48), ENTER SERIAL NUMBER FOR NEXT EMPLOYEE RECORD TO BE INSERTED, is superimposed - left justified, on line 3 of the display.
- The user enters another serial number. The DISPLAY service verifies it as described for step 2a. When the serial number passes the verification tests, the DISPLAY service returns control to the function.
- 7b** If the return code is 0, the display operation is successfully completed. Go to step 3a to verify that no record already exists for this employee number.
- 7c** If the return code is 8, the END or RETURN command was entered on the display by the user. Go to step 8a to end processing.
- 8a** Close the table TAB1: write it from virtual storage to permanent storage.
- 8b** End the function.



---

| EMP SER | 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 44. Five Rows in Table Library Member TAB1 (TAB1 is Referenced by Steps 1b, 3a, 6a, and 8a)

---

```

)BODY
%----- EMPLOYEE SERIAL -----
%COMMAND ==>_ZCMD
+
%ENTER EMPLOYEE SERIAL BELOW:
+
+
+   EMPLOYEE SERIAL%==>_EMP SER+   (MUST BE 6 NUMERIC DIGITS)
+
+
+PRESS%ENTER+TO DISPLAY NEXT SCREEN FOR ENTRY OF EMPLOYEE DATA.
+
+PRESS%END KEY+(PF 3) TO END THIS SESSION.

)PROC
  VER (&EMP SER, NONBLANK, PICT, NNNNNN)

)END

```

Figure 45. Panel Library Member, Panel Definition SER (Used in Steps 2a, 4a, and 7a)

---

---

----- EMPLOYEE SERIAL -----  
COMMAND ==>

ENTER EMPLOYEE SERIAL BELOW:

EMPLOYEE SERIAL ==>\_ (MUST BE 6 NUMERIC DIGITS)

PRESS ENTER TO DISPLAY NEXT SCREEN FOR ENTRY OF EMPLOYEE DATA.

PRESS END KEY (PF 3) TO END THIS SESSION.

Figure 46. Panel Display SER (Displayed by Steps 2a, 4a, and 7a)

---

---

----- EMPLOYEE SERIAL ----- ENTER REQUIRED FIELD  
COMMAND ===>

ENTER EMPLOYEE SERIAL BELOW:

EMPLOYEE SERIAL ===>\_ (MUST BE 6 NUMERIC DIGITS)

PRESS ENTER TO DISPLAY NEXT SCREEN FOR ENTRY OF EMPLOYEE DATA.

PRESS END KEY (PF 3) TO END THIS SESSION.

Figure 47. Panel Display SER With an ISPF-provided Message Superimposed on Line 1  
(Displayable During Steps 2a and 7a)

---

EMPX210 'DUPLICATE NUMBER' .ALARM=YES  
'EMPLOYEE RECORD ALREADY EXISTS FOR THIS NUMBER. ENTER ANOTHER.'

EMPX211 'NEW RECORD INSERTED'  
'ENTER SERIAL NUMBER FOR NEXT EMPLOYEE RECORD TO BE INSERTED.'

EMPX212 'ENTER PHONE NUMBER'  
'IF THE EMPLOYEE HAS NO PHONE, ENTER 000-0000.'

EMPX213 'ENTER FIRST NAME'  
'A FIRST NAME OR FIRST INITIAL IS REQUIRED.'

EMPX214 'ENTER LAST NAME'  
'A LAST NAME IS REQUIRED.'

Figure 48. Message Library Member EMPX21 (Used by Steps 4a, 5b, and 7a)

---

----- EMPLOYEE SERIAL ----- DUPLICATE NUMBER  
COMMAND ==>

ENTER EMPLOYEE SERIAL BELOW:

EMPLOYEE SERIAL ==> 598304 (MUST BE 6 NUMERIC DIGITS)

PRESS ENTER TO DISPLAY NEXT SCREEN FOR ENTRY OF EMPLOYEE DATA.

PRESS END KEY (PF 3) TO END THIS SESSION.

Figure 49. Panel Display SER With the Short Form of Message EMPX210 Superimposed on Line 1 (Displayed by Step 4a)

---

---

----- EMPLOYEE SERIAL ----- DUPLICATE NUMBER  
COMMAND ==>  
EMPLOYEE RECORD ALREADY EXISTS FOR THIS NUMBER. ENTER ANOTHER.  
ENTER EMPLOYEE SERIAL BELOW:

EMPLOYEE SERIAL ==> 598304 (MUST BE 6 NUMERIC DIGITS)

PRESS ENTER TO DISPLAY NEXT SCREEN FOR ENTRY OF EMPLOYEE DATA.

PRESS END KEY (PF 3) TO END THIS SESSION.

Figure 50. Panel Display SER With the Long Form of Message EMPX210 Superimposed on Line 3 (Displayable During Step 4a)

---

---

```

)BODY
%----- EMPLOYEE RECORDS -----
%COMMAND ==> _ZCMD
+
%   EMPLOYEE SERIAL: &EMP SER
+
+   EMPLOYEE NAME:
+   LAST   %=> _LNAME      +
+   FIRST  %=> _FNAME      +
+   INITIAL%=> _I+
+
+   HOME PHONE:
+   AREA CODE   %=> _PHA+
+   LOCAL NUMBER%=> _PHNUM  +
+
+
+PRESS%ENTER+TO STORE EMPLOYEE DATA AS ENTERED ABOVE.
+
+PRESS%END KEY+(PF 3) TO END THIS SESSION.

)INIT
  .CURSOR = LNAME
  IF (&PHA = ' ')
    &PHA = 914

)PROC
  VER (&LNAME,ALPHA)
  VER (&FNAME,ALPHA)
  VER (&I,ALPHA)
  VER (&PHA,NONBLANK,PICT,NNN)
  VER (&PHNUM,PICT,'NNN-NNNN')
  VER (&LNAME,NONBLANK,MSG=EMPX214)
  VER (&FNAME,NONBLANK,MSG=EMPX213)
  VER (&PHNUM,NONBLANK,MSG=EMPX212)

)END

```

Figure 51. Panel Library Member, Panel Definition DATA (Used in Step 5b)

---

---

----- EMPLOYEE RECORDS -----

COMMAND ==>

EMPLOYEE SERIAL: 106085

EMPLOYEE NAME:

LAST ==> \_

FIRST ==>

INITIAL ==>

HOME PHONE:

AREA CODE ==>

LOCAL NUMBER ==>

PRESS ENTER TO STORE EMPLOYEE DATA AS ENTERED ABOVE.

PRESS END KEY (PF 3) TO END THIS SESSION.

Figure 52. Panel Display DATA (Displayed by Step 5b)

---

## APPENDIX B. USING THE ISPF PARMS OPTION

The Parms option allows a user to display and change a variety of ISPF parameters at any time during the ISPF session. Changes remain in effect until the user changes the parameter again, and are saved from session to session. The parameter options menu is shown in Figure 53.

---

```
----- ISPF PARAMETER OPTIONS -----  
OPTION ==> _  
  
1  TERMINAL  - Specify terminal characteristics  
2  LOG/LIST  - Specify ISPF log and list defaults  
3  PF KEYS   - Specify PF keys for 3277 terminal with 12 PF keys
```

Figure 53. Parameter Options Menu

---

### SPECIFY TERMINAL CHARACTERISTICS (OPTION 0.1)

When a user selects this option, a panel is displayed that allows him to specify the terminal type, number of program function (PF) keys, the default pad character for panel input fields, the mode of operation for a 3278 Model 5, and the command stacking delimiter.

For MVS and VM/SP, the initial defaults are shown in Figure 54.



For VSE, the initial defaults are shown in Figure 55.

The allowable alternatives for these defaults are indicated on the display.

After review and any changes are made to these parameters, enter the END command to return to the previous menu.

---

```
----- TERMINAL CHARACTERISTICS -----
COMMAND ==> _

TERMINAL TYPE    ==> 3277_ (3277 - 3275/3277 terminal)
                  (3277A - 3275/3277 with APL keyboard)
                  (3278 - 3276/3278/3279 terminal)
                  (3278A - 3276/3278/3279 with APL keyboard)
                  (3278T - 3276/3278/3279 with TEXT keyboard)

NUMBER OF PF KEYS ==> 12 (12 or 24)

INPUT FIELD PAD  ==> N (N - Nulls)
                  (B - Blanks)

SCREEN FORMAT    ==> DATA (DATA - Format based on data width)
(3278 Model 5 only) (STD - Always format 24 lines by 80 chars)
                  (MAX - Always format 27 lines by 132 chars)

COMMAND DELIMITER ==> ; (Special character for command stacking)
```

Figure 54. Terminal Characteristics Panel (MVS and VM/SP)

---

---

```

----- TERMINAL CHARACTERISTICS -----
COMMAND ==> _

TERMINAL TYPE    ==> 3277_ (3277 - 3275/3277 terminal)
                  (3277A - 3275/3277 with APL keyboard)
                  (3278 - 3276/3278/3279 terminal)
                  (3278A - 3276/3278/3279 with APL keyboard)
                  (3278T - 3276/3278/3279 with TEXT keyboard)

NUMBER OF PF KEYS ==> 12 (12 or 24)

INPUT FIELD PAD  ==> N (N - Nulls)
                  (B - Blanks)

OPTIMIZE DISPLAY ==> N (Y - Optimize display data)
                  (N - No optimization of display data)

COMMAND DELIMITER ==> ; (Special character for command stacking)

```

Figure 55. Terminal Characteristics Panel (VSE)

---

Specification of terminal type allows ISPF to recognize valid (displayable) characters. A 3278 terminal can display six more characters than a 3277. If you have a 3279 terminal, specify 3278 as the terminal type, since a 3279 terminal has the same character set as a 3278.

**Note:** One or more of following installation-dependent options for terminal type may also be included on this panel:

```

3277KN - for 3277 Katakana terminals
3278CF - for 3278 Canadian French terminals
3278KN - for 3278 Katakana terminals

```

Specification of the number of PF keys controls the particular set of PF key definitions currently in use, and also affects the panel displayed by option 0.3.

In the following cases, ISPF automatically senses the terminal type and number of PF keys:

- If the screen size is greater than 24 lines (determined when the user logs on), ISPF sets the terminal type to 3278.
- If the user presses a PF key higher than 12, ISPF sets the terminal type to 3278 and the number of PF keys to 24.

ISPF cannot sense the terminal type or number of PF keys in the following cases:

- If the user switched between a 3277 and 3278 Model 2 (both 24-line terminals).
- If the user switched from a terminal with 24 PF keys to a terminal with 12 PF keys.

In these cases, the user must inform ISPF of the terminal type and number of PF keys using option 0.1 or 0.3. Otherwise, incorrect character set and PF key definitions will be used (see option 0.3).

ISPF automatically determines the terminal type during ISPF initialization, and sets it to the appropriate value.

ISPF automatically sets (or changes) the number of PF keys in the following cases:

- If the terminal type is 3277, ISPF initializes the number of keys to 12.
- If the terminal type is 3278, ISPF initializes the number of keys to whatever was "remembered" from the user's last ISPF session (for a new user, the number of keys is initialized to 12).
- If the user presses a PF key higher than 12, ISPF sets the number of keys to 24.

ISPF cannot sense the number of PF keys if a user has switched from a 3278 with 24 PF keys to a 3278 with 12 PF keys. In this case, the user must inform ISPF of the number of PF keys through option 0.1 (Terminal Characteristics). Otherwise, the incorrect set of "remembered" key definitions will be used (see option 0.3).

Specification of a pad character controls the initial padding of panel input fields (including selection panels) but not the data portion of an edit display. Within edit, null or blank padding is controlled with edit commands.

Specification of display optimization controls the amount of data written to the terminal during each display operation. Specify N to have the complete screen image written to the terminal for each display request. This option is appropriate for terminals that are locally attached to the processor. Specify Y to have only changes to the screen

image written to the terminal. This option is appropriate for terminals remotely attached to the processor.

Users can stack commands on the command line by separating them with a delimiter. The default delimiter, the semicolon, may be changed using this option. Stacking allows the user to enter, for example:

```
====> FIND DEPT;HEX ON
```

which finds the characters DEPT and then displays the file at that point in hex mode.

## MVS: SPECIFY LOG AND LIST DEFAULTS (OPTION 0.2)

When an MVS user selects this option, a panel (Figure 56) is displayed that allows him to specify default processing for log and list files, lines per page, and allocation parameters, to be used when he terminates ISPF using primary option X.

---

```
----- LOG AND LIST DEFAULTS -----
COMMAND ==>

LOG DATA SET DEFAULT OPTIONS          LIST DATA SET DEFAULT OPTIONS
-----                                -----
Process option   ==> J_                 Process option   ==> J
SYSOUT class    ==> A                   SYSOUT class    ==> A
Local printer ID ==>                    Local printer ID ==>
Lines per page  ==> 60                  Lines per page  ==> 60
Primary pages   ==> 10                  Primary pages   ==> 100
Secondary pages ==> 10                  Secondary pages ==> 100

VALID PROCESS OPTIONS:
  J - Submit job to print (and delete)      K - Keep data set (do not print)
  L - Route to local printer (and delete)   D - Delete data set (do not print)

JOB STATEMENT INFORMATION: (If option "J" selected)
====> //HOSTETLA JOB (U602,B043),'HOSTETLER RS',NOTIFY=HOSTETL
====>
====>
====>
```

Figure 56. Log and List Defaults Panel (MVS)

The initial defaults are:

|                 | Log File | List File |
|-----------------|----------|-----------|
| SYSOUT Class    | ====> A  | ====> A   |
| Lines per Page  | ====> 60 | ====> 60  |
| Primary Pages   | ====> 10 | ====> 100 |
| Secondary Pages | ====> 10 | ====> 200 |

No defaults are supplied for the other parameters on this panel.

Normal values for lines per page are:

60 - for printing 6 lines per inch  
80 - for printing 8 lines per inch

Primary/secondary allocation parameters are specified in terms of anticipated number of pages of printout. These values are automatically converted by ISPF to the appropriate number of blocks prior to allocating space for the log and list files.

If the user modifies the primary/secondary allocation parameters after the files have been allocated, the new values take effect the next time you enter ISPF. (The list file is allocated the first time the user requests a print function. The log file is allocated the first time a user performs some action that results in a log message, such as saving edited data or submitting a batch job.)

For the log file, the user may specify a primary allocation of 0 (zero) to prevent allocation and generation of the log. Users can avoid allocating the list file by simply not requesting any print functions.

If a user requests default processing options for the log and list files, the following rules apply:

- If printing as a batch job is specified, SYSOUT class and job statement information must also be specified. (If option J for both log and list is specified, different SYSOUT classes may be specified but only one job is submitted for printing both files.)
- If routing to a local printer is specified, a printer id must also be specified.

If these rules are not observed, or if no default processing options are specified, primary option X or the RETURN command causes the termination menu to be displayed.

After reviewing or changing the parameters on this panel, enter the END command to return to the previous menu.

## VM/SP: SPECIFY CONSOLE, LOG, AND LIST DEFAULTS (OPTION 0.2)

When a VM/SP user selects this option, a panel (Figure 57) is displayed that allows him to specify default processing for the virtual console and for the log and list files, to be used when he terminates ISPF using primary option X. If the user has not specified default processing options, primary option X will cause the ISPF termination menu to be displayed.

---

```
----- CONSOLE, LOG, AND LIST DEFAULTS -----
COMMAND ==>

CONSOLE PROCESS OPTION ==> - (K or D)          LINES PER PAGE:
LOG PROCESS OPTION      ==> (P, K, D, or N)      LOG ==> 80
LIST PROCESS OPTION     ==> (P, K or D)         LIST ==> 80

VALID PROCESS OPTIONS:
  P - Print file (and delete)
  K - Keep file - do not print
  D - Delete (purge) file - do not print
  N - Do not generate log file

LOG/LIST SPOOL OPTIONS:
  NUMBER OF COPIES ==> 1          SPOOL CLASS ==> A
  BIN NUMBER       ==>          'FOR' USER ==>
  3800 KEYWORDS   ==>

FOR SPOOLING LOG/LIST TO ANOTHER PERSON OR MACHINE:
  USER/MACHINE ID ==>
  NODE/LINK ID     ==>
  TAG TEXT         ==>
```

Figure 57. Console, Log, and List Defaults Panel (VM/SP)

---

Users may also specify the number of lines per page and spool parameters for printing the log and list files. The initial defaults are:

```
Lines per page ==> 80 (for both log and list)
Number of copies ==> 1
Spool class    ==> A
```

No ISPF-supplied defaults are provided for the other parameters on this panel.

Normal values for lines per page are:

- 60 - for printing 6 lines per inch
- 80 - for printing 8 lines per inch

The virtual console is automatically started the first time ISPF invokes a CMS command (typically, a LINK or ACCESS executed automatically) Upon termination of ISPF, the console may be kept (in "start" status) or deleted (purged and set to "stop" status). An appropriate processing option for the console would be:

**K** - for users who normally run with a virtual console.

**D** - for users who normally run without a virtual console.

The ISPF log file is created the first time the user performs some action which results in a log message, such as saving edited data or submitting a job to the batch machine. The ISPF list file is created the first time a user requests a print function.

For the log file, users may specify a process option of N to prevent generation of the log. Users may avoid generation of the list file by simply not requesting any print functions.

See Chapter 5, "Invocation and Termination" for a discussion of spool parameters that may be specified for the log and list files.

After reviewing or changing the parameters on this panel, enter the END command to return to the previous menu.

## VSE: SPECIFY LOG AND LIST DEFAULTS (OPTION 0.2)

When a VSE user selects this option, a panel (Figure 58) is displayed that allows him to specify default processing for the log and list files, lines per page, and POWER JECL statements to be used when he terminates ISPF using primary option X.

The initial defaults are:

|                | Log File | List File |
|----------------|----------|-----------|
| Lines per Page | ==> 60   | ==> 60    |

Normal values for lines per page are:

- 60 - for printing 6 lines per inch
- 80 - for printing 8 lines per inch

If a user requests default processing option J (print via a batch job), the POWER JECL statement information must be specified.

---

```

----- LOG AND LIST DEFAULTS -----
COMMAND ==>

LOG DATA SET DEFAULT OPTIONS          LIST DATA SET DEFAULT OPTIONS
-----                                -----
Process option   ==> _                 Process option   ==>
Lines per page  ==> 60                 Lines per page   ==> 60

VALID PROCESS OPTIONS FOR LOG AND LIST DATA SETS:
  J - Submit job to print (and delete)      K - Keep data set (do not print)
  P - Write to ICCF print area and          D - Delete data set (do not print)
       display on terminal (and delete)

POWER JECL STATEMENTS: (If option "J" selected)
==> * $$ JOB JNM=ZLWGA
==> *
==> *
==> *

```

Figure 58. Log and List Defaults Panel (VSE)

---

**Note:** Option D means that the data in the data set is not reused. The physical data set is not deleted.

If option K is requested, ISPF attempts to extend the log or list data set in the next session, unless the data set was defined with a /FILE statement specifying DISP=D.

**Note:** Extending the data set is accomplished by saving the next available record location for the log and list data sets in the system profile table ISPSPROF. This information is used, in the next session, to position to the next available record in the log or list data set.

Thus, there is a relationship between the system profile table ISPSPROF and the physical location of the log and list data sets. Should the log or list data set starting location not match the information in the profile table or should the next available record address not be within the current log or list data set, that data set will be written from the beginning, just as though DISP=K had not been specified in the previous session.



Sharing the same log or list data set between different users is not permitted because this could lead to conflicting or erroneous information being placed in the system profile table.

If option P is selected, ISPF prints the data set using system logical unit SYSLST. This causes ICCF to display the data on the terminal and to write the data to the ICCF \$\$PRINT area.

If default processing options are not specified, primary option X or the RETURN command causes the termination menu to be displayed.

After reviewing or changing the parameters on this panel, enter the END command to return to the previous menu.

### SPECIFY PROGRAM FUNCTION KEYS (OPTION 0.3)

The PF key definition panel allows users to assign PF keys to ISPF commands. A user may assign PF keys to system commands (such as HELP or END), to commands that are meaningful within a particular function or environment (such as the edit FIND and CHANGE commands), and to line commands (such as edit or dialog test I or D commands).

When entering the KEYS command or select option 3 from the ISPFParms option menu, the panel shown in Figure 59 is displayed.

The PF key definitions shown in the figure are the default definitions distributed with ISPF.

Before changing PF key assignments, verify the terminal type and the number of PF keys (12 or 24). The terminal type must be one of the following:

3277, 3277A, 3277KN, 3278, 3278A, 3278CF, 3278KN, 3278T

The panel shown in Figure 59 is the panel that is displayed for terminals with 12 PF keys. For terminals with 24 PF keys, the first panel displayed by the KEYS command or by option 0.3 shows the "primary" keys (PF13-PF24). When the ENTER key is pressed, a panel is displayed showing the "alternate" keys (PF1-PF12). Flip-flop between the two panels by continuing to press ENTER. See Figure 60.

Users can define or change a PF key function simply by equating the key to a command. Example:

```
PF9 ==> CHANGE ALL ABC XYZ
PF12 ==> PRINT
```

In the example, PF9 has been equated to an edit command, and PF12 has been equated to the system-defined PRINT command.

---

```
----- PF KEY DEFINITION -----
COMMAND ==> _
NUMBER OF PF KEYS ==> 12          TERMINAL TYPE ==> 3278

PF1 ==> HELP
PF2 ==> SPLIT
PF3 ==> END
PF4 ==> RETURN
PF5 ==> RFIND
PF6 ==> RCHANGE
PF7 ==> UP
PF8 ==> DOWN
PF9 ==> SWAP
PF10 ==> LEFT
PF11 ==> RIGHT
PF12 ==> CURSOR
```

INSTRUCTIONS:

Verify number of PF keys and terminal type before proceeding.  
Press ENTER key to process changes.  
Enter END command to process changes and exit.

Figure 59. PF Key Definition Panel (12 PF Keys)

---

If a blank is entered for any PF key definition, the key is restored to its ISPF default. The defaults are discussed under "Program Function Keys."

When a PF key definition begins with a colon, it indicates a line command. The colon is stripped off and the command to which the key is equated is inserted in the first input field in the line at which the cursor is currently positioned.

When a PF key definition begins with a greater-than sign, the command is passed through to the dialog via the command field. The command table is not searched. This is provided for compatibility with the previous SPF product.

---

----- PF KEY DEFINITION - PRIMARY KEYS -----

COMMAND ==>

NUMBER OF PF KEYS ==> 24

TERMINAL TYPE ==> 3278

PF13 ==> HELP  
PF14 ==> SPLIT  
PF15 ==> END  
PF16 ==> RETURN  
PF17 ==> RFIND  
PF18 ==> RCHANGE  
PF19 ==> UP  
PF20 ==> DOWN  
PF21 ==> SWAP  
PF22 ==> LEFT  
PF23 ==> RIGHT  
PF24 ==> PRINT

INSTRUCTIONS:

Verify number of PF keys and terminal type before proceeding.  
Press ENTER key to process changes and display alternate keys.  
Enter END command to process changes and exit.

Figure 60 (Part 1 of 2). PF Key Definition Panels (24 PF Keys)

---

---

----- PF KEY DEFINITION - ALTERNATE KEYS -----  
COMMAND ==>

NOTE: The definitions below apply only to terminals with 24 PF keys.

PF1 ==> HELP  
PF2 ==> SPLIT  
PF3 ==> END  
PF4 ==> RETURN  
PF5 ==> RFIND  
PF6 ==> RCHANGE  
PF7 ==> UP  
PF8 ==> DOWN  
PF9 ==> SWAP  
PF10 ==> LEFT  
PF11 ==> RIGHT  
PF12 ==> CURSOR

INSTRUCTIONS:

Press ENTER key to process changes and display primary keys.  
Enter END command to process changes and exit.

Figure 60 (Part 2 of 2). PF Key Definition Panels (24 PF Keys)

---



## APPENDIX C. USING THE TBDISPL SERVICE

This appendix describes the use of the TBDISPL and TBPOT services in a dialog function that displays rows of a table for possible modification by a user.

During function processing, the TBDISPL service is used to control displays of a table, TAB1. Changes the user desires to make in TAB1, are entered on the display directly in fields of displayed lines. Each field corresponds to a table variable. Each line corresponds to a row of the table.

After the user enters changes to a line or lines of the display, the function, by using TBDISPL, locates each line changed by the user and positions the table current row pointer (CRP) at the row in the table that corresponds to the line. TBDISPL operation then stores the contents of this row and values from the changed line of the display in function pool variables. Finally, the function uses the TBPOT service to write the updated function pool variables to the table row.

A user may originate a function command while viewing the displayed panel. Logic is included that checks for this eventuality.

The user ends function processing by entering the END or RETURN command (or by pressing the END or RETURN PF key) on the panel displayed by the function.

Logic that deletes or inserts rows in the table is not included in the function. Also, not illustrated is the capability to perform verification of user-enter information, in conjunction with panel display, through specifications on the panel definition. That is illustrated in Appendix A, "Using the DISPLAY Service"; see processing description step 5b.

(The function may be initiated by a user at a terminal by means of the ISPSTART command. If the user has already started ISPF, the function may be initiated, at the terminal, from a menu (on which the function is invoked by one of the provided selections) or from a display, containing a command line, by means of SELECT action in a command table. Or, the function may be initiated by another function using the SELECT service.)

Steps in dialog function processing are listed on the next page. Each step has a step identifier (1a, 2a, 2b, etc.). This identifier refers to a description, on the page following, of the processing performed by the step.

## STEPS IN FUNCTION PROCESSING

This section lists the steps in the function.

Additional comments about function processing follow in "Description of Steps in Function Processing."

**Note:** Dialog service requests in the steps (TBOPEN, TBDISPL, etc.), are in the command procedure format. Program format dialog service requests are illustrated with each service description in Chapter 6. Additional examples of services requests in both the command procedure (CLIST, EXEC) and program (COBOL, FORTRAN, PL/I) format may be found in ISPF Dialog Management Services Examples.

| <u>Step</u><br><u>Id</u> | <u>Function Processing</u>      | <u>Comment</u>                                                            |
|--------------------------|---------------------------------|---------------------------------------------------------------------------|
| 1a.                      | TBOPEN TAB1 WRITE               | Open table TAB1.                                                          |
| 2a.                      | TBDISPL TAB1 PANEL(T1PANEL)     | Display table TAB1.                                                       |
| 2b.                      | if return code = 0, go to 3a    | 0 = one line modified and/or command entered.                             |
| 2c.                      | if return code = 4, go to 4a    | 4 = two or more lines modified. A command may have been entered, as well. |
| 2d.                      | if return code = 8, go to 5a    | 8 = END or RETURN entered.                                                |
| 3a.                      | TBQUERY TAB1 POSITION(CHECKCRP) | If CRP=0, scrollable line was not changed,                                |
| 3b.                      | if CHECKCRP = 0, go to 7a       | but command field (ZCMD) was changed.                                     |
| 3c.                      | TBPUT TAB1                      | Process final line on this screen.                                        |
| 3d.                      | if ZCMD is not null, go to 7a   | If not null, a value is in the command field.                             |
| 3e.                      | if ZCMD is null, go to 2a       | If null, redisplay the table.                                             |
| 4a.                      | TBPUT TAB1                      | Process one of two or more lines remaining.                               |
| 4b.                      | TBDISPL TAB1                    |                                                                           |
| 4c.                      | if return code = 0, go to 3a    | 0 = one line and/or a command remain.                                     |
| 4d.                      | if return code = 4, go to 4a    | 4 = two or more lines remain to be processed.                             |
| 5a.                      | TBQUERY TAB1 POSITION(CHECKCRP) | User entered END or RETURN command on the                                 |
| 5b.                      | if CHECKCRP = 0, go to 6a       | screen just displayed. Process any lines                                  |
| 5c.                      | TBPUT TAB1                      | changed on this screen and then end the                                   |
| 5d.                      | TBDISPL TAB1                    | function.                                                                 |
| 5e.                      | go to 5a                        |                                                                           |
| 6a.                      | TBCLOSE TAB1                    | End the function.                                                         |
| 6b.                      | End the function                |                                                                           |
| 7a.                      | process function command        | Function command processing                                               |
| 7b.                      | CONTROL DISPLAY SAVE            | - Save screen contents and status.                                        |
|                          | .                               | - Process the function command, including                                 |
|                          | .                               | any displays.                                                             |
| 7c.                      | CONTROL DISPLAY RESTORE         | - Restore screen contents and status                                      |
|                          |                                 | that were stored in step 7b.                                              |
| 7d.                      | go to 2a                        | Redisplay the table.                                                      |

## DESCRIPTION OF STEPS IN FUNCTION PROCESSING

The description below is related, by step id, to "Steps in Function Processing" above. Referenced figures are collected at the end of the description.

| <u>Step Id</u> | <u>Description</u>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1a             | Open the table: read table TAB1 (contents of which are shown in Figure 61) into virtual storage for update.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 2a             | Display table TAB1 (beginning with the row at which the CRP is positioned). The display, as it appears at the terminal, is shown in Figure 62. Format of the display is controlled by a panel definition named T1PANEL (shown in Figure 63). TBDISPL, in addition to displaying the table, allows the user to scroll up and down the scrollable data in the display.<br><br>The user may continue, indefinitely, to scroll through the displayed table. Control will be returned to the function when the user does one of the following: <ul style="list-style-type: none"><li>• enters the END or RETURN command without entering data on the panel.</li><li>• enters a change in one or more lines of scrollable data and/or enters data in the command field (which is variable ZCMD, shown in Figure 63) and presses the ENTER key, enters a SCROLL command, or enters the END or RETURN command.</li></ul><br>When a line in the scollable part of the display has been changed, TBDISPL retrieves - from the table - the row corresponding to that line (i.e., the row values are stored in the function pool). The table CRP is positioned at this row when the row is retrieved. Next, values from the changed line are stored in the function pool. If no lines are changed, the CRP is set to zero. Any value entered in the command field is stored in variable ZCMD. |
| 2b             | If the return code is 0, a single line was changed by the user; or an entry made in the command field (ZCMD) by the user, was passed through to this function; then the user pressed the ENTER key or entered a SCROLL command. Go to step 3a to process the line and/or the value in the command field.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 2c             | If the return code is 4, more than one line was changed and the ENTER key was pressed or a SCROLL command was entered by the user. Go to step 4a to update the table.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |



- 2d** If the return code is 8, the END or RETURN command was entered by the user, go to step 5a to determine if any lines were changed.
- 3a** Use TBQUERY to obtain the position of the CRP in a variable named CHECKCRP.
- 3b** If the CRP is 0, no scrollable lines were changed by the user, but the command field (ZCMD) contains data that was originated by the user while viewing the display.
- 3c** This TBPUR writes values stored in the function pool to the table row.
- A single line was changed by the user or several lines were changed and the line currently being processed is the final one to be processed.
- 3d** When ZCMD is not null, it contains data that was originated by the user while viewing the display.
- 3e** Go to step 2a to redisplay the table, positioning the most recently changed line at the top of the display.
- 4a** This TBPUR writes values stored in the function pool to the table row.
- More than one line was changed by the user.
- 4b** A TBDISPL request issued with no panel name and message-id specified, positions the CRP to the row corresponding to the next line of the screen in which the application user made a change. That row is retrieved (i.e., stored in the function pool). Next, values from the line are stored in the function pool.
- 4c** Go to step 3a if a single line remains to be processed.
- 4d** Go to step 4a if more than one line remains to be processed.
- 5a** The END or RETURN command was entered by the application user. Save the position of the CRP in a variable named CHECKCRP.
- 5b** If the CRP is 0, no changed lines remain to be processed for the just displayed screen, or the END or RETURN command was issued by the user without having changed any lines.

(The processing logic ignores any function command originated by the application user in conjunction with the END or RETURN command. The function developer, however, may choose to accept for processing function commands entered with END or RETURN. Function command processing is described by steps 3d and 7a through 7c.)

- 5c** If the CRP is nonzero, update the row of the table at which the CRP is positioned.
- 5d** A TBDISPL request issued with no panel name and message-id specified, positions the CRP to the row corresponding to the next line of the screen in which the application user made a change. That row is retrieved (i.e., stored in the function pool). Next, values from the line are stored in the function pool.
- (Because the END or RETURN command has been entered by the user, a return code of 8 is returned each time for this TBDISPL request.)
- 5e** Go to 5a to test the CRP for 0. If 0, no more lines containing changes remain to be processed.
- 6a** Close the table: write table TAB1, as revised, from virtual storage to permanent storage.
- 6b** End the function.
- 7a** Process the data, in ZCMD - the command field, originated by the user. (This data is either a bona fide function command - i.e., data anticipated by function processing - or invalid data entered by the user).
- 7b** If function command processing is to include any BROWSE, EDIT, DISPLAY, or TBDISPL operations, use CONTROL DISPLAY SAVE to save the contents and status of the currently displayed screen.
- 7c** CONTROL DISPLAY RESTORE restores the screen previously saved by CONTROL DISPLAY SAVE so that processing can be resumed.
- If non-ISPF displays are processed, instead of using CONTROL DISPLAY SAVE and RESTORE, use CONTROL DISPLAY REFRESH either before or after the non-ISPF display is done.
- 7d** Go to 2a to resume processing by redisplaying the table.

---

| EMP SER | 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 61. Table TAB1 Contents

---

```

----- EMPLOYEE LIST ----- LINE 000001 COL 001 080
COMMAND INPUT ==> _ SCROLL ==> PAGE

ENTER NECESSARY REVISIONS (OTHER THAN CHANGES TO EMPLOYEE SERIAL):

----- EMPLOYEE NAME -----          --- PHONE ---          EMPLOYEE
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
***** BOTTOM OF DATA *****

```

Figure 62. Table TAB1 as Displayed

---

---

```

)ATTR
  _ TYPE(INPUT) INTENS(LOW)
  @ TYPE(OUTPUT) INTENS(LOW)
)BODY
%----- EMPLOYEE LIST -----
%COMMAND INPUT ==>_ZCMD                                %SCROLL ==>_AMT +
%
+ENTER NECESSARY REVISIONS (OTHER THAN CHANGES TO EMPLOYEE SERIAL):
+
+----- EMPLOYEE NAME -----          --- PHONE ---      EMPLOYEE
+LAST          FIRST          MI          AREA  NUMBER      SERIAL
+
)MODEL
_LNAME          _FNAME          _I          _PHA  _PHNUM          _EMPSER+
)INIT
  &AMT = PAGE
)END

```

Figure 63. Table Display Panel Definition T1PANEL

---



## APPENDIX D. COMMAND TABLE UTILITY

*does not work in  
"TEST" mode*

The command table utility is a component of ISPF. It may be invoked as option 1 from the sample primary option menu (ISP@PRIM) distributed with ISPF. It may also be invoked as a utility from the PDF primary option menu (option 3.9) if PDF is installed. Use of this utility allows command tables to be generated or modified.

The first panel displayed by this utility prompts the user for an application id. See Figure 64. The name of the command table is derived by appending "CMDS" to the application id. If the table exists in the table input library, it is displayed and may be modified. If the table does not exist in the table input library, a new table is generated.

**Note:** This utility cannot be used to modify a command table that is currently in use. Command table ISPCMDS (the system command table) is always in use by ISPF. To modify this table, the user must make a copy of the table, rename the copy, modify the copy, and replace the original with the copy outside the ISPF environment.

---

```
----- COMMAND TABLE UTILITY -----
COMMAND ==>

ENTER/VERIFY APPLICATION ID BELOW:
APPLICATION ID ==> _

The name of the command table to be processed is formed by prefixing
the application id to the string 'CMDS'. For example:
APPLICATION ID ==> TST
results in a command table name of 'TSTCMDS'.
```

Figure 64. Command Table Utility Panel

---

The second panel displayed by the command table utility is shown in Figure 65. For each row of the table, it shows the command verb, the truncation amount (T), the action, and the description. The description is displayed on a separate line, offset under the action.

For a new table, this panel initially contains a full screen of dummy entries in which all fields are displayed with underscores. The underscores are pad characters, and need not be blanked out.

The entries in this display may be scrolled up and down, and one or more entries may be modified simply by overtyping. In addition, the following line commands may be entered at the left of any entry by overtyping the four quote marks:

- I or In - insert one or "n" lines. The inserted line(s) will contain underscores (pad characters) in all field positions.
- R or Rn - repeat one or "n" lines. The repeated line(s) will contain underscores (pad characters) in the verb and truncation fields, but the action and description fields will be replicated from the line on which the "R" or "Rn" was entered.

- D or Dn - delete one or "n" lines.

In addition to the two scroll commands (UP and DOWN), the following primary commands are supported:

- END - causes the table to be saved in the table output library, and terminates the utility.
- CANCEL - terminates without saving the table. May be abbreviated "CAN".

**Notes:**

1. Multiple line commands or modifications may be entered in a single interaction. The lines are processed in the order in which they appear on the screen.
2. Any line commands or modifications that are entered concurrently with the END command are processed before the table is saved.
3. Any null entries (in which at least the verb contains all underscores) are automatically deleted when the table is saved.

```
COMMAND TABLE - TSTCMDS ----- LINE 000001 COL 001 080
COMMAND ==>                               SCROLL ==> PAGE
```

INSERT, DELETE, AND CHANGE COMMAND ENTRIES. UNDERSCORES NEED NOT BE BLANKED.  
 ENTER END COMMAND TO SAVE CHANGES OR CANCEL TO END WITHOUT SAVING.

| VERB                       | T | ACTION<br>DESCRIPTION                                               |
|----------------------------|---|---------------------------------------------------------------------|
| '''' SORT                  | 0 | SELECT PGM(PQRSORT) PARM(&ZPARM)<br>SORT ENTRIES BY ASCENDING ORDER |
| '''' PREPARE               | 4 | SELECT CMD(XPREP &ZPARM) NEWPOOL<br>PREPARE FILE FOR FORMATTING     |
| '''' QUIT                  | 2 | ALIAS END<br>QUIT COMMAND - SAME FUNCTION AS END                    |
| '''' EXPLAIN               | 4 | ALIAS HELP<br>EXPLAIN COMMAND - SAME FUNCTION AS HELP               |
| '''' UP                    | 0 | &SCRVERT<br>SCROLL UP COMMAND                                       |
| '''' DOWN                  | 0 | &SCRVERT<br>SCROLL DOWN COMMAND                                     |
| ***** BOTTOM OF DATA ***** |   |                                                                     |

Figure 65. Command Table Editing Panel





## APPENDIX E. SUMMARY OF ISPF SYNTAX

This appendix contains a quick reference of syntax for invoking an ISPF application, message definitions, skeleton control statements, panel definitions, and dialog service requests.

### INVOKING AN ISPF APPLICATION

(See page 88)

```
ISPSTART { PANEL(panel-name) [OPT(option)]  
          { CMD(command)  
            { PGM(program-name) [PARM(parameters)]  
              { [LANG(PLI|PL1) [,storage-area]]  
                { [LANG(COBOL)]  
                  [NEWAPPL[(application-id)]]  
                  [TEST|TESTX|TRACE|TRACEX]  
                  [NOABEXIT]
```

### MESSAGE DEFINITIONS

(See page 251)

```
msgid ['short message'] [.HELP = panel-name|*] [.ALARM = YES|NO]  
'long message'
```

### SKELETON CONTROL STATEMENTS

(See page 256)

```
)DEFAULT abcdefg  
)TB value1 ... value8  
)IM skel-name [NT] [OPT]  
)SEL relational-expression  
)ENDSEL  
)DOT table-name  
)ENDDOT
```

)SET variable = expression

)CM comment

## PANEL DEFINITIONS

### Panel Header Statements

All parameters on header statements are optional. When preparing a panel header statement, use only one line.

#### Attribute Section

(Optional section, see page 206)

)ATTR [DEFAULT (abc | %+\_)]

#### Body Section

(Required section, see page 210)

)BODY [CMD(field-name)] [DEFAULT(abc | %+\_)]  
[MSG(field-name)] [KANA]  
[LMSG(field-name)]

#### Model Section *for TABLE DISPLAY panels only*

(Optional section, see page 244)

)MODEL [CLEAR(var-name,var-name...)] [ROWS(ALL|SCAN)]

#### Initialization Section

(Optional section, see page 213)

)INIT

#### Processing Section

(Optional section, see page 213)

)PROC

## Statement Specifying the End of a Panel Definition

(Required statement, see page 199)

)END

## Panel Statements and Built-in Functions

### Attribute Section

One or more parameters are required with each attribute character specified. These parameters are (see page 207):

|                                           |                       |
|-------------------------------------------|-----------------------|
| TYPE(TEXT  <u>I</u> NP <u>U</u> T OUTPUT) | CAPS( <u>O</u> N OFF) |
| INTENS(HIGH LOW NON)                      | PAD(NULLS char)       |
| JUST( <u>L</u> EFT RIGHT ASIS)            | ATTN( <u>O</u> N OFF) |
| SKIP( <u>O</u> N OFF)                     |                       |

### Initialization and Processing Sections

(See page 214)

variable = value

TRUNC (variable,value)

TRANS (variable value,value...[MSG=message-id])

IF (variable operator value[,value...])

VPUT name-list [ASIS|SHARED|PROFILE]

VER (variable[,NONBLANK],keyword[,value...][,MSG=message-id])

### Panel Control Variables

(See page 223)

.CURSOR = field-name

.RESP = ENTER|END

.HELP = panel-name

.TRAIL = variable

*ALARM = {YES}*

```
.MSG      = message-id
.ZVARS    = '(name-list)'
```

## DIALOG SERVICES

The command invocation syntax (for CLIST and EXEC 2) for all services is shown first, followed by the call invocation syntax for PL/I. For other language syntax, refer to "Call Invocation" in Chapter 6. The services are described in Chapter 6 in alphabetic sequence.

### Command Invocation Syntax

Refer to Chapter 6 for complete description of the services, where they appear in alphabetic sequence.

#### Display Services

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

ISPEXEC  TBDISPL  table-name [PANEL(panel-name)]
              [MSG(message-id)]
              [CURSOR(field-name)]
              [CSRROW(table-row-number)]

ISPEXEC  SETMSG   MSG(message-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 [LIBRARY(library-name)]
              [NEWCOPY|REPLCOPY]
              [NAME(alt-name)]
              [PAD(percentage)]
```

|         |         |            |                                                                                        |
|---------|---------|------------|----------------------------------------------------------------------------------------|
| ISPEXEC | TBCLOSE | table-name | [LIBRARY(library-name)]<br>[NEWCOPY REPLCOPY]<br>[NAME(alt-name)]<br>[PAD(percentage)] |
| ISPEXEC | TBEND   | table-name |                                                                                        |
| ISPEXEC | TBERASE | table-name | [LIBRARY(library-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)]<br>[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)]<br>[SAVENAME(var-name)]     |
| ISPEXEC | TBV CLEAR | table-name |                                              |

#### File Tailoring Services

|         |         |                     |                                     |
|---------|---------|---------------------|-------------------------------------|
| ISPEXEC | FTOPEN  | [TEMP]              |                                     |
| ISPEXEC | FTINCL  | skel-name           | [NOFT]                              |
| ISPEXEC | FTCLOSE | [NAME(member-name)] | [LIBRARY(library-name)]<br>[NOREPL] |
| ISPEXEC | FTERASE | member-name         | [LIBRARY(library-name)]             |

## Variable Services

ISPEXEC VGET name-list [ASIS|SHARED|PROFILE]

ISPEXEC VPUT name-list [ASIS|SHARED|PROFILE]

## Other Services

ISPEXEC SELECT { PANEL(panel-name) [OPT(option)]  
                  CMD(command)  
                  PGM(program-name) [PARM(parameters)]  
                  [NEWAPPL[(application-id)]|NEWPOOL] }

ISPEXEC CONTROL { DISPLAY { LOCK  
                              LINE [START(line-number)]  
                              SM [START(line-number)]  
                              REFRESH  
                              SAVE|RESTORE }  
                  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(message-id)

## Call Invocation Syntax

Refer to Chapter 6 for complete description of the services, where they appear in alphabetic sequence.

### Display Services

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

CALL ISPLINK ('TBDISPL',    table-name [,panel-name]
                [,message-id]
                [,field-name]
                [,table-row-number] );

CALL ISPLINK ('SETMSG',     message-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]
                [,library-name]);

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

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

CALL ISPLINK ('TBERASE',   table-name [,library-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] [,library-name]
                                [, 'NOREPL']);
CALL ISPLINK ('FTERASE', member-name [,library-name]);
```

## Variable Services

```
CALL ISPLINK ('VGET',      name-list [, 'ASIS' | 'SHARED' | 'PROFILE' ] );
CALL ISPLINK ('VPUT',      name-list [, 'ASIS' | 'SHARED' | 'PROFILE' ] );
CALL ISPLINK ('VDEFINE',   name-list, variable, format, length
                        [, options-list] [, user-data] );
CALL ISPLINK ('VDELETE',   name-list);
CALL ISPLINK ('VCOPY',      name-list, array-1, array-2
                        [, 'LOCATE' | 'MOVE' ] );
CALL ISPLINK ('VREPLACE',  name-list, lengths, values);
CALL ISPLINK ('VRESET');
```

## Other Services

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

Note: parameters which may appear in buffer are:

```
      {
        PANEL(panel-name) [OPT(option)]
        CMD(command)
        PGM(program-name) [PARM(parameters)]
                          [LANG(PLI|PL1 [,storage-area])]
                          [LANG(COBOL)]
        [NEWAPPL [(application-id)]|NEWPOOL]
      }
```

```
CALL ISPLINK ('CONTROL',   type [,mode]
                        [,line-number] );
CALL ISPLINK ('BROWSE',    dsname [,serial]
                        [,pswd-value] );
CALL ISPLINK ('BROWSE',    fileid [,member-name] );
CALL ISPLINK ('BROWSE',    dsname [,recfm]
                        [,lrecl]
                        [,blksize]
                        [,sysno]
                        [,volser] );
CALL ISPLINK ('EDIT',      dsname [,serial]
                        [,pswd-value] );
CALL ISPLINK ('EDIT',      fileid [,member-name] );
```

```
CALL ISPLINK ('EDIT',    dsname [,profile]
                    [,recfm]
                    [,lrecl]
                    [,blksize]
                    [,sysno]
                    [,volser] );

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

## APPENDIX F. VDEFINE EXIT ROUTINE

The dialog writer may specify an exit routine to define dialog variables when program variables are nonstandard (other than CHAR, FIXED, BIT, or HEX). Then, when a variable is accessed by any ISPF service, the exit routine is invoked to perform any conversion necessary between the program variable's format and the character string format required for a dialog variable.

This type of exit is specified with the format of USER. A data area must be supplied that contains the address of the exit subroutine along with any other user data. If the defined variable name is '\*', all unresolved dialog variable accesses result in invocation of the exit routine. (Unresolved dialog variables are those which were not implicitly entered or defined in the function pool.)

The exit routine is invoked by a call (BALR 14,15) and standard OS linkage conventions must be followed. When a variable read is being performed a return code of 8 by the user exit indicates that the variable was not found. All other nonzero return codes for either read or write requests are considered severe errors.

The parameters are a request flag, data length, data address, defined storage length, defined storage address, and the user data area. The exit is invoked with:

```
CALL XRTN( UDATA,      /* invoke exit and pass user area */
          SRVCODE,    /* request code                      */
          NAMESTR,    /* name length and name chars        */
          DEFLEN,     /* defined area length                */
          DEFAREA,    /* defined area                       */
          SPFDLEN,    /* ISPF data length                  */
          SPFDATAP); /* ISPF data address                 */
          /* to ISPF data on read request */
```

### UDATA

An area that follows the exit routine address parameter, specified on the VDEFINE statement. This area may contain any additional information the user desires. Its format is CHAR(\*).

### SRVCODE

Service request-type code, as a fullword fixed value. The allowable values are 0 for a read and 1 for a write. Other values should be accepted without error, in order to allow further extensions. (Codes of 2 and 3 are used by the dialog test facility variable query function. Code 2 is a request for the number of variables to be returned in the SPFDLEN field. Code 3 is a request for the names of the variables to be returned in the buffer pointed to by SPFDATAP. The names are entered as contiguous 8-byte tokens.)

**NAMESTR**

Name of the dialog variable being requested, preceded by the one-byte name length.

**DEFLEN**

The length of the area specified to the VDEFINE service. Its format is a fullword fixed value.

**DEFAREA**

The area specified to the VDEFINE service. Its format is CHAR(\*);

**SPFDLEN**

For a write request, the length of the SPFDATA area is supplied. For a read request, the length of the data is returned to ISPF. It must be supplied by the exit routine. Its format is a fullword fixed value.

**SPFDATAP**

For a write request, the address of the data to be stored is supplied. For a read request, the address of the data is returned to ISPF. Its format is a fullword pointer.

The following return codes are possible and should be set in the exit routine:

- 0 - Successful operation
- 8 - Variable not found on read request
- Others - Severe error

## APPENDIX G. CHARACTER TRANSLATIONS FOR APL, TEXT, AND KATAKANA

ISPF permits use of APL keyboards for all models of 3270 terminals, and text keyboards for 3278 and 3279 terminals. The 2-byte transmission codes for APL and text characters are translated by ISPF into 1-byte codes for internal storage as shown in Figure 66 and Figure 67.

ISPF also permits use of 3277 and 3278 Japanese Katakana terminals. The character codes are documented in IBM 3270 hardware manuals. Many of the Katakana codes overlay the lowercase EBCDIC codes. In a panel definition, it is assumed that lowercase EBCDIC characters are to be displayed for these codes, unless the )BODY header statement includes the keyword KANA. Example:

```
)BODY KANA
```

The keyword, KANA, is used on a )BODY header statement when Katakana characters are included within the panel. When KANA is specified, rules for display of text fields are as follows (input and output fields and model line fields are not affected by use of the KANA keyword):

- If the terminal type is Katakana, and
  - the KANA keyword is present, text characters are left as is.
  - the KANA keyword is not present, any lowercase text characters are translated to uppercase and uppercase text characters are left as is.
- If the terminal type is not Katakana, and
  - the KANA keyword is present, any lowercase text characters are treated as being non-displayable and are translated to a period. Any uppercase text characters are left as is.
  - the KANA keyword is not present, lowercase and uppercase text characters are left as is.

|    | 0  | 1        | 2        | 3        | 4        | 5        | 6        | 7        | 8        | 9        | A | B  | C | D | E | F |    |
|----|----|----------|----------|----------|----------|----------|----------|----------|----------|----------|---|----|---|---|---|---|----|
| 00 |    |          |          |          |          |          |          |          |          |          |   |    |   |   |   |   | 0F |
| 10 |    |          |          |          |          |          |          |          |          |          |   |    |   |   |   |   | 1F |
| 20 |    |          |          |          |          |          |          |          |          |          |   |    |   |   |   |   | 2F |
| 30 |    |          |          |          |          |          |          |          |          |          |   |    |   |   |   |   | 3F |
| 40 | sp | <u>A</u> | <u>B</u> | <u>C</u> | <u>D</u> | <u>E</u> | <u>F</u> | <u>G</u> | <u>H</u> | <u>I</u> | ¢ | .  | < | ( | + |   | 4F |
| 50 | &  | <u>J</u> | <u>K</u> | <u>L</u> | <u>M</u> | <u>N</u> | <u>O</u> | <u>P</u> | <u>Q</u> | <u>R</u> | ! | \$ | * | ) | ; | ¬ | 5F |
| 60 | -  | /        | <u>S</u> | <u>I</u> | <u>U</u> | <u>V</u> | <u>W</u> | <u>X</u> | <u>Y</u> | <u>Z</u> |   | ,  | % | _ | > | ? | 6F |
| 70 |    | ^        | ..       |          |          |          |          | v        | '        | :        | # | @  | ' | = | " |   | 7F |
| 80 | ~  | a        | b        | c        | d        | e        | f        | g        | h        | i        | ↑ | ↓  | ≤ | Γ | L | → | 8F |
| 90 | □  | j        | k        | l        | m        | n        | o        | p        | q        | r        | ⊃ | ⊂  |   | o |   | ← | 9F |
| A0 | -  | ~        | s        | t        | u        | v        | w        | x        | y        | z        | ∩ | ∪  | ⊥ | [ | ≥ | ° | AF |
| B0 | α  | ε        | ι        | ρ        | ω        |          | x        | \        | ÷        |          | ∇ | Δ  | τ | ] | ≠ |   | BF |
| C0 | {  | A        | B        | C        | D        | E        | F        | G        | H        | I        | ∞ | ∞  |   | φ |   | ⊙ | CF |
| D0 | }  | J        | K        | L        | M        | N        | O        | P        | Q        | R        | I | !  | ∇ | Δ | ⊞ | ⊞ | DF |
| E0 | \  |          | S        | T        | U        | V        | W        | X        | Y        | Z        | + | +  |   | e | ⊞ | ⊞ | EF |
| F0 | 0  | 1        | 2        | 3        | 4        | 5        | 6        | 7        | 8        | 9        |   | ∇  | Δ | ⊙ | ⊙ |   | FF |

 3278 only; invalid character on 3277.


 National use character. Graphics shown are for U.S. keyboards; graphics differ in other countries.

Figure 66. Internal Character Representations for APL Keyboards

|    | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B  | C | D | E | F |  |    |
|----|----|---|---|---|---|---|---|---|---|---|---|----|---|---|---|---|--|----|
| 00 |    |   |   |   |   |   |   |   |   |   |   |    |   |   |   |   |  | 0F |
| 10 |    |   |   |   |   |   |   |   |   |   |   |    |   |   |   |   |  | 1F |
| 20 |    |   |   |   |   |   |   |   |   |   |   |    |   |   |   |   |  | 2F |
| 30 |    |   |   |   |   |   |   |   |   |   |   |    |   |   |   |   |  | 3F |
| 40 | sp |   |   |   |   |   |   |   |   |   | ¢ | .  | < | ( | + |   |  | 4F |
| 50 | &  | 1 | 2 | 3 |   |   |   |   |   | ↓ | ! | \$ | * | ) | ; | ~ |  | 5F |
| 60 | -  | / |   |   |   |   |   |   |   |   | ! | ,  | % | _ | > | ? |  | 6F |
| 70 | n  | ° |   |   |   |   |   |   |   | ' | : | #  | @ | ' | = | " |  | 7F |
| 80 |    | a | b | c | d | e | f | g | h | i | ↑ | {  | ≤ | ( | + | + |  | 8F |
| 90 | □  | j | k | l | m | n | o | p | q | r |   | }  | ≠ | ) | ± | ■ |  | 9F |
| A0 | -  | ~ | s | t | u | v | w | x | y | z | • | L  | Γ | [ | ≥ | • |  | AF |
| B0 | °  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ∇ | ┘  | ┘ | ] | ≠ | - |  | BF |
| C0 | {  | A | B | C | D | E | F | G | H | I | Δ | ⊥  | τ | v | + | + |  | CF |
| D0 | }  | J | K | L | M | N | O | P | Q | R | ⊞ | Δ  | § | ¶ | ← | → |  | DF |
| E0 | \  | \ | S | T | U | V | W | X | Y | Z | ∇ | †  | † | ▲ | ▼ | ∧ |  | EF |
| F0 | 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |   | L  | φ | φ | Γ |   |  | FF |


 National use character. Graphics shown are for U.S. keyboards; graphics differ in other countries.

Figure 67. Internal Character Representations for Text Keyboards





## APPENDIX H. MVS AND VM/SP: SUMMARY OF CHANGES FROM SPF

For MVS and VM/SP, the dialog management functions of ISPF and the program development functions of ISPF/PDF were previously combined in the predecessor program product, System Productivity Facility (SPF). While some functions have been added and some SPF functions revised in the ISPF product, many SPF dialogs can run, without change, on ISPF. New and revised function in ISPF that may require changes in an SPF dialog to be processed using ISPF, are listed at the end of this section.

### NEW AND REVISED FUNCTIONS

New and revised ISPF functions are listed below.

- Invoke ISPF using the ISPSTART command instead of the ISPF command.
- Name-List Syntax

A name-list parameter in a CALL ISPLINK need not be enclosed in parentheses when there is only one variable in the list.

- NEWAPPL Keyword

The SELECT service and the ISPSTART command allow specification of an optional application id on the NEWAPPL keyword. For example:

```
SELECT PANEL(ABCTOP) NEWAPPL(XXXX)
```

where XXXX is the application id.

The application id, which must be unique to the individual application, may be up to four characters. It serves as a prefix to identify the user's application profile and/or command table associated with the application, as follows:

```
XXXXPROF - User Application Profile  
XXXXEDIT - Edit Profile  
XXXXCMDS - Command Table
```

The NEWAPPL keyword does not automatically designate a menu as being a primary option menu. To designate a menu as a primary option menu, set -- in the )INIT section of menu -- the &ZPRIM system variable to "YES."

- NOCHECK Keyword

A NOCHECK keyword may be specified with either the CMD or PGM keyword on selection panels. With this keyword, the dialog function (command or program) is invoked even if the user specifies a chain of options.

- Variable Services

- Profile variables are included in standard search sequence.
- A name-list interface is provided for the VCOPY and VREPLACE services.
- User exits are provided for variable access and conversion.

- Command Tables

A facility is included that allows the dialog manager to intercept user-entered commands and take appropriate action.

- Program Function (PF) Key

- Program function keys are not required for operation of ISPF. All functions for which PF keys were required for SPF may now be entered in the command field of any display.
- All panels may now have a command entry field, which is also used for option entry on menus.
- PF keys may now be used to simulate command entry.
- All previous PF key functions (HELP, SPLIT, END, etc.) may now be entered as commands.
- The FIND and CHANGE PF key functions have been renamed RFIND (repeat find) and RCHANGE (repeat change) to avoid confusion with the FIND and CHANGE commands passed through to browse and edit.

- Light Pen and Cursor Select

Fields on a panel can be specified to be light pen detectable or detectable via the cursor select key.

- Set Next Selection Menu

A system variable, ZPARENT, allows the developer to indicate the next panel to be displayed when the user presses the end key or when a function completes operation. This allows a dialog developer to control the sequence of menu display, if he should require it.

- Display Services

- A set next message function provides the ability to specify a message to be displayed with the next panel that is written by ISPF to the terminal.
- The table display services have been enhanced with the following features:
  - Multiple line selection or modification
  - New )MODEL header statement keywords
  - Variable names on model lines
  - Multiple model lines
  - Explicit cursor placement within scrollable data
- Enhancements to the panel display service include:
  - Extended verification functions
  - "Z" variables as field name placeholders
  - Truncation remainder function
  - Alternate locations for command and message fields
  - Profile variable support
  - New attribute keywords (SKIP and ATTN)
  - KANA keyword for Japanese keyboards
- Help panels may now contain variables so that dialog information (including information entered by the user) may be displayed on the help panel.
- Control Services
  - An option has been added to allow display output without unlocking the keyboard.
  - Split screen mode may be enabled or disabled by a function.
  - The panel and table display environments may be saved and restored.
- Batch Execution of Dialog Services
 

Non-interactive dialogs may be executed in the background.
- IPF Table Migration Utility (VM/SP only)
 

A utility is provided to assist with the conversion of tables.

- APL and TEXT keyboards may now be used. However, although the terminals are capable of displaying a mixture of characters from both keyboards at one time, only those characters appropriate for the keyboard in use are displayed by ISPF.
- New commands:
  - TSO** (MVS only) allows a TSO command or CLIST to be entered from any panel.
  - CP, CMS** (VM/SP only) allows CP or CMS commands or EXECs to be entered from any panel.
  - PANELID** Used on a panel, allows all subsequently displayed panels to include, in their display, the name of the specified panel.
- Specify PF Keys (Option 0.3)
 

The PF key specification is changed. For 24-key terminals, two panels are now displayed: The first for specification of "primary" keys (13-24), and the second for specification of "alternate" keys (1-12).
- The command table utility is provided for generation and modification of application command tables.
- A selection panel update utility is provided, which adds selections to existing selection panels. Use of this utility is described in ISPF and ISPF/PDF Installation and Customization.

## MIGRATION OF DIALOGS FROM SPF TO ISPF

Differences between ISPF and SPF may require that some dialogs, written to run under SPF, be modified if they are to run under ISPF:

- Changes to default variable pool setting of the VGET/VPUT services may cause a different variable value to be obtained when in ISPF.
- Additional VGET/VPUT Return codes.
- SPF dialogs that use EDIT/BROWSE interfaces will require that PDF be installed to execute under ISPF.

Command tables should be reviewed if a dialog is to be invoked from a new dialog which uses command tables. The 'new' dialog may have defined commands in the command table that conflict with commands that were processed by the dialog function routine or panels of the SPF dialog.

ISPPARMS does not exist in ISPF. It is replaced by ISPPROF in a different format. A conversion process is described in ISPF and ISPF/PDF Installation and Customization.

ISPF primary option menus are no longer determined by the NEWAPPL keyword on a SELECT request. To identify a panel as a primary option menu, set the variable ZPRIM to YES in the INIT section of the panel definition.

Tables created using SPF can be processed by ISPF. However, tables created using ISPF cannot be processed by SPF. After an SPF table is processed in WRITE mode by ISPF, it can not be processed by SPF.

Programs that ran on SPF and which call ISPLINK, must be link edited again to use the ISPF version of the ISPLINK module.

Although not required it is recommended that:

- Panels using OPT and SEL (SPF variable names) be converted to use ISPF variable names ZCMD and ZSEL, respectively.
- Help/tutorial panels be converted to use ZUP and ZCONT in place of UP and CONT. Variable ZIND be set to YES in the PROC section of tutorial index pages. This will cause a higher level tutorial text page to be displayed when the user enters the UP command in response to a tutorial page that was displayed as a result of a selection from the index instead of redisplaying the index page as was done in SPF. If ZIND is not set to YES in the index page, the tutorial function will continue to work as it did in SPF.



## APPENDIX I. VM/SP: USE OF SHARED MINIDISKS

ISPF permits multiple users to have concurrent write access to the same minidisks (shared minidisk support). Thus multiple users may concurrently create or update different files, library members, or tables on a shared minidisk without destructive conflicts. ISPF ensures the integrity of shared minidisks if updating of them is done via ISPF services. However, ISPF 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:

- Do not attempt to use any user's 191 disk (A-disk) as a shared minidisk; ISPF does not provide shared disk support for such minidisks.
- Isolate shared user files, ISPF tables, and file tailoring output files on minidisks that do not have other types of files.
- Caution users to update shared minidisks only by using ISPF services.
- Control LINKing and ACCESSing:
  - Allow ISPF to dynamically link and access shared minidisks. This is done automatically for ISPF libraries, and may be specifically requested in the appropriate panel fields for native CMS files.
  - LINK and ACCESS shared minidisks using VM/CMS services, but always access them as read-only extensions. This avoids inadvertent updating (e.g., by some compilers). For example:

```
CP LINK TO BLVIT 291 AS 591 MW
ACCESS 591 D/D
```

In this example, the disk is linked in multiple write mode, thus providing for concurrent updating by multiple users. However, since the disk is accessed as a read-only extension of itself (D/D), writing to the disk is prevented, except when under ISPF control. ISPF automatically reaccesses the disk when it needs to write to the disk, and then restores the original access mode (in this case, D/D) when the writing is completed.

Whenever ISPF updates a shared minidisk, it serializes all simultaneous update requests for the same minidisk by multiple users via a locking mechanism. Additionally, when ISPF is finished writing to a minidisk for a given user, but before ISPF "unlocks" the minidisk, it checks to see if the user has any other minidisks currently accessed (i.e., as another mode letter) which are in fact the same minidisk as the one



being currently updated by that user. If so, every such "redundant" minidisk will be reaccessed so as to keep its in-memory CMS master file directory current.

## INDEX

### Special Characters

.CURSOR control variable  
description 223  
example  
    see figure on page 225  
    when not initialized or set to  
    blank 226

.HELP control variable 239  
description 223  
example  
    see figure on page 225, 237  
    in message definition 252

.MSG control variable  
description 223

.RESP control variable  
description 224

.TRAIL control variable  
description 224  
example 215, 231

.ZVARS control variable 226  
description 224  
example  
    see figure on page 227

&PRESUME statement 105

)CM file tailoring control  
statement 258

)DEFAULT skeleton control statement 256

)DOT file tailoring control  
statement 257  
example 31

)END statement required on panel  
definition 203

)ENDDOT file tailoring control  
statement 257  
example 31

)ENDSEL file tailoring control  
statement 257  
example 30

)IM file tailoring control  
statement 256

)SEL file tailoring control  
statement 257  
example 30

)SET file tailoring control  
statement 258

)TB  
    file tailoring control statement 256

% sign  
    beginning a command with 89, 121,  
    141

### A

abbreviated (generic) search argument  
    See TBSARG  
    See TBSCAN

abend intercept  
    VSE 95

access skeleton files (FTOPEN) 137

accessing table data 23

add row to table (TBADD) 147

ALIAS action  
    specified via ZCTACT 54

aliases  
    for commands 56

allocating libraries  
    MVS 66  
    VM/SP 69

ALPHA parameter  
    on VER statement 220

alt-name parameter  
    on TBCLOSE 150  
    on TBSAVE 177

APL keyboards  
    character translations 313

application  
    definition of 38

application commands  
    description 45

application profile pool  
    creation of 38  
    definition of 33  
    description of 38  
    life of 38

application-id parameter  
    on ISPSTART 87  
    on SELECT 140

ARGLIST(name-list) parameter  
    on TBSARG 176  
    on TBSCAN 180

- ASIS parameter
  - on VGET 192
  - on VPUT 194
  - on VPUT statement in panel )INIT and )PROC sections 223
  - with JUST keyword 208
- assembler language
  - VL keyword 106
- assignment statement
  - in panel definition
    - )INIT section 214
    - )PROC section 214
- attention exits, CLIST 104
- attention field
  - command entry 45, 47
  - selection 47
- attention key (PA1)
  - MVS and VM/SP 62
  - VSE 63
- ATTN exits, CLIST 104
- ATTN Keyword
  - in )ATTR section 210
  - in panel )ATTR section 207
  - used to define attention fields 63
- attribute characters 207
- attribute section of panel
  - definition 199, 206
- automatic and non-automatic entry into
  - line mode 121, 141

B

- BACK tutorial command 16, 239
- batch environment
  - TSO 95
  - VM/SP 98
  - VSE 100
- batch execution 95
  - TSO error processing 96
  - TSO sample job 96
  - VM/SP error processing 98
  - VM/SP sample job 98
  - VSE error processing 101
  - VSE sample job 100
- beginning ISPF 87
- BIT parameter
  - on VDEFINE 189
  - on VER statement 220
- blksize parameter
  - on BROWSE
    - VSE 116

- on EDIT
  - VSE 129
- body section of panel definition 199, 210
- BROWSE and EDIT service 42
- BROWSE service
  - description
    - MVS and VM/SP 113
    - VSE 116
  - example
    - MVS and VM/SP 115
    - VSE 119
- browse services panel definition
  - scroll field, location of 201
- bypassing display
  - See jump function

C

- calculation of DTCB size 28
- calculation of table size 28
- call invocation 105
  - general format 106
  - positional parameters 106
- CANCEL mode
  - effect on error processing 109
- CANCEL parameter
  - on CONTROL 120
- CAPS keyword
  - in panel )ATTR section 207, 208
- change row in table
  - TBMOD 167
  - TBPUT 171
- changes, differences between ISPF and SPF 317
- changing number of lines to be scrolled
  - See scrolling amount
- CHAR parameter
  - on VDEFINE 189
  - with PAD keyword 209
- character translations for APL, TEXT and Katakana keyboards 313
- check variable content 219
- checking the content of panel input
  - fields (use of the VER statement) 219
- clear table variables to nulls
  - (TBVCLEAR) 185
- CLIST libraries
  - MVS 67
- CLIST variables used in commands 104
- close and save table (TBCLOSE) 150

close table without saving (TBEND) 162

CM file tailoring control statement  
See )CM

CMD parameter  
on panel body section 211

CMD(command) parameter  
in )PROC section 230  
on ISPSTART 89  
on SELECT 140

CMS system command 49

COBOL  
return codes from services  
MVS and VM/SP 110  
VSE 111

COBOL, compiler used with ISPF 1

column display position  
table display operations 201

column of a table, defining 22

command entry  
concatenated with PF key  
restriction 47

command field  
key-in of command entries 45  
naming via the CMD parameter 211  
panel body section 211  
position in panel definition 201

command field, naming of 201

command invocation 103  
general format 103  
positional parameters 104

command parameter  
in panel )PROC section 230

command procedure function pool 35

command processing restriction  
split screen mode 48  
with HELP command 46

command stacking  
restriction with HELP command 46

command table utility 297

command tables 53  
action commands 54  
dynamically specified command  
actions 58  
format of 53  
modification of 297  
used for assigning command  
aliases 56  
used for overriding system  
commands 56  
used for passing commands to a dialog  
function 57

commands  
entry of 45  
interception by ISPF 45  
levels of 45  
processing of  
by ISPF 46  
stacking for execution 46  
system  
See system commands

comment statements 204

communication between functions and  
panels via shared pool 38

compare variable content 219

compilers used with ISPF 1

concatenation of PF key entered value  
with command entry  
restriction 47

console default for VM/SP  
See also specifying options using  
parms  
setting using parms 281

CONT system variable  
on tutorial panels 240

control facilities 12

control of scrolling  
See CSR

CONTROL service  
description 43, 120  
error disposition  
when CANCEL specified 109  
when RETURN specified 109  
example 123, 263, 290

control statements  
skeleton definition 254

control variables 223  
example  
see figure on page 225  
initialization 224  
list of 223  
when reset 224

copy a variable (VCOPY) 186

COPY parameter  
on VDEFINE 190

copy variables to shared pool or profile  
(VPUT) 194

CP system command 49

create a new table (TBCREATE) 153

creating a table  
example 26

creating dialog components 3

creation of application profile  
pools 38

creation of shared pools 38

CRP (current row pointer) 24

crp-name parameter  
on TBQUERY 173

CRP, movement of

- See TBBOTTOM
- See TBSCAN
- See TBSKIP
- See TBTOP
- CSRROW(table-row-number) parameter
  - on TBDISPL 157
- CUR
  - scrolling amount 52
- current-row-pointer
  - See CRP
- CURSOR (.CURSOR) control variable
  - See .CURSOR
- cursor control of scrolling
  - See CUR
- cursor positioning
  - default 226
  - example 226
- cursor select 63
  - processing of selected fields 63
- cursor select key
  - selection of attention field 47
- CURSOR system command 48
- CURSOR(field-name) parameter
  - on DISPLAY 124

D

- data communications within ISPF 7
- data records
  - in skeleton definition 254
- data table control block (DTCB)
  - calculation of size 28
- DATASET(dsname) parameter
  - on BROWSE
    - MVS and VM/SP 113
    - VSE 116
  - on EDIT
    - MVS and VM/SP 126
    - VSE 129
- default PF Key assignments 59
- DEFAULT skeleton control statement
  - See )DEFAULT
- define a table search argument
  - (TBSARG) 175
- define function variable
  - VDEFINE 188
- defined variables 36
- defining display images 3
- defining primary option menus 232
- delete (set to nulls) table values
  - (TBVCLEAR) 185

- delete a table (TBERASE) 163
- delete row from table (TBDELETE) 155
- determine if row exists in table
  - (TBEXIST) 164
- determining table size 28
- DEVADDR(sysno) parameter
  - on EDIT
    - VSE 129
- dialog
  - beginning with menu or function 5, 9
  - control concepts 9
  - cross-system use 1
  - definition of 1
  - development of 3
    - MVS and VM/SP 72
    - VSE/AF 1.3.5 78
    - VSE/AF 2.1 85
  - example 261, 289
  - initiation of 12, 93
  - invocation of
    - using application master menu 6
  - organization concepts 9
  - processing concepts 9
  - recursive entry into 56
  - running of 5
  - services overview 17
  - termination of 93
  - typical organization 9
- dialog abend intercept
  - VSE 95
- dialog components
  - creation of 3, 8
  - test of 3
    - MVS and VM/SP 72
    - VSE/AF 1.3.5 78
    - VSE/AF 2.1 85
- dialog development libraries
  - MVS and VM/sp 72
  - VSE/AF 1.3.5 78
  - VSE/AF 2.1 85
- dialog function
  - languages used for coding 1
- dialog variable
  - See variables
- differences between ISPF and SPF 317
- DISABLE parameter
  - on CONTROL 120
- display bypassing
  - See jump function
- display data set
  - MVS and VM/SP (BROWSE) 113
  - VSE (BROWSE) 116
- display file
  - MVS and VM/SP(BROWSE) 113

VSE(BROWSE) 116  
display menu (SELECT) 140  
DISPLAY parameter  
  on CONTROL 120  
DISPLAY service  
  description 17, 124  
  example 27, 125, 261, 263  
display services 17  
  syntax summary - calls 307  
  syntax summary - commands 304  
display table data (TBDISPL) 156  
display, set the next message to  
  (SETMSG) 145  
display, specification of 3  
DOT file tailoring control statement  
  See )DOT  
DOWN scroll command 52  
DOWN system command 47  
dsname parameter  
  on BROWSE  
    MVS and VM/SP 113  
    VSE 116  
  on EDIT  
    MVS and VM/SP 126  
    VSE 129  
  on VER statement 221  
DTCB  
  calculation of size 28  
dynamically specified command  
  actions 58

E

EDIT and BROWSE service 42  
EDIT service  
  description  
    MVS and VM/SP 126  
    VSE 129  
  example  
    MVS and VM/SP 128  
    VSE 132  
edit services panel definition  
  scroll field, location of 201  
editing the content of panel input  
  fields (use of the VER statement) 219  
ENABLE parameter  
  on CONTROL 120  
end file tailoring (FTCLOSE) 133  
END parameter

  on CONTROL 120  
END statement 199  
END system command 47  
  See also RETURN system command  
  distinguishing from RETURN 51  
  processing 50  
ENDDOT file tailoring control statement  
  See )ENDDOT  
ending display  
  See END system command  
ending function or dialog  
  See END system command  
  See RETURN system command  
ending ISPF 93  
ENDSEL file tailoring control statement  
  See )ENDSEL  
ENQ issued by TBOPEN 169  
enqueueing for table write operations 25  
ENTER parameter  
  on CONTROL 120  
entry of commands  
  See commands  
ENTRY statement required when link  
  editing VSE programs 112  
environment 1  
erase (set to nulls) table variables  
  (TBVCLEAR) 185  
erase a table (TBERASE) 163  
erase member of file tailoring output  
  library (FTERASE) 135  
error modes  
  CANCEL and RETURN 109  
error processing  
  TSO batch execution 96  
  VM/SP batch execution 98  
  when put into effect 93  
ERRORS parameter  
  on CONTROL 120  
examples of menus 234  
EXEC variables used in commands 104  
EXEC 2 libraries, VM  
  SP 71  
EXIT parameter  
  in )PROC section 230, 231  
exit routine  
  VDEFINE service 311  
explicit chain mode 233  
extended return  
  See jump function  
extension variables  
  table 24

F

field selection via cursor position 63  
 field selection via light pen 63  
 field type specification  
   in panel )ATTR section 208  
 field-name parameter  
   on DISPLAY 124  
   on TDBISPL 157  
 file tailoring end (FTCLOSE) 133  
 file tailoring libraries  
   MVS 66  
   VM/SP 69  
   VSE 78, 84  
 file tailoring services  
   description 28  
   example 31  
   output 28  
   processing 28  
     in conjunction with table  
     processing 29  
   skeleton  
     example 29  
   syntax summary - calls 308  
   syntax summary - commands 305  
 FILE(fileid) parameter  
   on BROWSE  
     MVS and VM/SP 113  
     VSE 116  
   on EDIT: MVS and EDIT 126  
 fileid parameter  
   on BROWSE  
     MVS and VM/SP 113  
   on EDIT  
     MVS and VM/SP 126  
   on VER statement 220  
 find table variable  
   TBSARG 175  
   TBSCAN 180  
 FIXED parameter  
   on VDEFINE 189  
 format  
   command invocation 103  
 format of skeleton control  
   statements 256  
 format of skeleton definition 254  
 format parameter  
   on VDEFINE 188  
 formula for table size calculation 28

FORTTRAN  
   return codes from services  
     MVS and VM/SP 110  
     VSE 111  
 FORTRAN, compiler used with ISPF 1  
 FTCLOSE service  
   description 133  
   example 31, 134  
 FTERASE service  
   description 135  
   example 135  
 FTINCL service  
   description 136  
   example 31, 136  
 FTOPEN service  
   description 137  
   example 31, 138  
 function  
   dialog  
     example 261, 289  
 function pool  
   creation of 35  
   definition of 33  
   description of 35  
   for command procedures 35  
   for programs 36  
   life of 35  
   relationship to dialog functions 35  
 function variables, define in function  
   pool (VDEFINE) 188  
 function, dialog  
   creation of 3  
   description of 7  
   languages used for coding 1

G

general table services 26  
 generic search argument  
   specification of  
     TBSCAN 180  
 generic search argument, specification  
   of  
     TBSARG 175  
 get a copy of variable (VCOPY) 186  
 get row from table (TBGET) 165  
 get variable from shared pool or profile  
   (VGET) 192

H

HALF  
    scrolling amount 52  
HELP (.HELP) control variable  
    See .HELP  
HELP command  
    entry to tutorial 16  
    stacking restriction 46  
HELP system command 47, 239  
help tutorial  
    special definition requirements 228  
HEX parameter  
    on VDEFINE 189  
HIGH parameter  
    with INTENS keyword 208

I

ICCF system command 48  
identical variable names 37  
IF statement  
    in panel )INIT section 218  
    in panel )PROC section 218  
IM file tailoring control statement  
    See )IM  
implicit variables 36  
include file tailoring skeleton  
    (FTINCL) 136  
INCLUDE statements required when link  
    editing VSE programs 112  
INDEX tutorial command 16, 239  
initialization of control variables 224  
initialization section of panel  
    definition 199, 213  
initiating a dialog 93  
input  
    protecting table 25  
INPUT parameter  
    with TYPE keyword 208  
INTENS keyword  
    in panel )ATTR section 207, 208  
interception of commands by ISPF 45  
introduction 1  
invocation of ISPF 87  
invoking a dialog (SELECT) 140  
invoking an ISPF application 5  
    from master application menu 6  
invoking SELECT service 93

invoking services 103  
ISPF differences with SPF 317  
ISPFIL ddname 66, 69  
ISPLINK 106  
ISPLINK routine, invoking services from  
    programs 2  
ISPLLIB ddname 67  
ISPLNK 106  
ISPLNK routine, invoking services from  
    FORTRAN programs 2  
ISPMLIB ddname 65, 68  
ISPPLIB ddname 65, 68  
ISPPROF ddname 65, 68  
ISPSLIB ddname 65, 68  
ISPSTART  
    description 87  
    example 5  
ISPSTART syntax 88  
ISPTABL ddname 66, 69  
ISPTLIB ddname 65, 66, 68, 69

J

JCL  
    allocating libraries for MVS 66  
job for batch execution  
    TSO 96  
    VM/SP 98  
    VSE 100  
jump function 51  
JUST keyword  
    in panel )ATTR section 207, 208

K

Katakana keyboards  
    character translations 313  
key-name parameter  
    on TBQUERY 173  
Key-name-list parameter  
    on TBCREATE 153  
KEYNUM(keynum-name) parameter  
    on TBQUERY 173  
KEYS system command 48  
    used for defining PF key  
    functions 60  
KEYS(key-name) parameter  
    on TBQUERY 173



KEYS(key-name-list) parameter  
on TBCREATE 153  
keyword parameter  
coding in requests for services 108

L

LANG(COBOL) parameter  
in panel )PROC section 230  
on ISPSTART 89  
on SELECT 140  
LANG(PLI) parameter  
in panel )PROC section 230  
on ISPSTART 89  
on SELECT 140  
languages used for coding functions 1  
LEFT parameter  
with JUST keyword 208  
LEFT scroll command 52  
LEFT system command 48  
length parameter  
on VDEFINE 190  
length-array parameter  
on VCOPY 186  
lengths parameter  
on VREPLACE 196  
levels of ISPF commands 45  
library setup  
MVS 65  
VM/SP 68  
VSE/AF 1.3.5 73  
VSE/AF 2.1 79  
LIBRARY(library-name) parameter  
on FTCLOSE 133  
on FTERASE 135  
on TBCLOSE 150  
on TBERASE 163  
on TBSAVE 177  
light pen 63  
processing of selected fields 63  
sample panel 64  
selection of attention field 47  
line mode  
automatic and non-automatic entry  
into line mode 121, 141  
LINE parameter  
on CONTROL 120  
lines  
number to be scrolled  
See scrolling amount  
link editing VSE programs

ENTRY statement required 112  
INCLUDE statements required 112  
linking requirement for split screen  
mode 68  
list file defaults  
specifying  
using parms option 275  
LIST parameter  
on VER statement 221  
LMSG parameter  
on panel body section 211  
LOCATE parameter  
on VCOPY 186  
LOCK parameter  
on CONTROL 120  
locks for table write operations 25  
log file defaults  
specifying, using parms option 279,  
281, 282  
LOG service  
description 43, 139  
example 139  
logging a message  
See LOG service  
logical screens  
See split screen mode  
long message 20  
LOW parameter  
with INTENS keyword 208  
lower-level menu  
example of 237  
lrecl parameter  
on BROWSE  
VSE 116  
on EDIT  
VSE 129

M

master application menu 6, 234  
example of 234  
MAX  
scrolling amount 52  
MEMBER(member-name) parameter  
on BROWSE  
MVS and VM/SP 113  
VSE 116  
on EDIT  
MVS and VM/SP 126  
member-name parameter  
on FTCLOSE 133

- on FTERASE 135
- menu
  - definition of primary option 232
  - entry to tutorial 16
  - example of primary option 236
  - lower-level
    - example of 237
  - master application
    - example 234
  - special definition requirements 228, 229
  - use of ZPARENT to set next display 233
- message
  - description of 7
  - set the display of next (SETMSG) 145
- message definition 199
  - creation of 20
  - in message library 21
  - long
    - example 22, 252
  - message-id 251
  - parts of 20
  - processing 251
  - short
    - example 22, 252
    - syntax 251
- message fields
  - panel body section 211
- message library 251
  - example 22, 252
- message logging
  - See LOG service
- message-id parameter
  - on DISPLAY 124
  - on LOG 139
  - on SETMSG 145
  - on TBDISPL 157
- minidisk, shared support 323
- model section of panel definition 199
- modes of operation
  - set by use of CONTROL service
    - CANCEL 109
    - RETURN 109
  - test 91
  - trace 93
- modify command tables 297
- modify row in table
  - TBMOD 167
  - TBPUT 171
- modifying displayed tables using TBDISPL 289
- modifying tables
  - example 261

- MOVE parameter
  - on VCOPY 186
- move row pointer (CRP)
  - See TBBOTTOM
  - See TBSCAN
  - See TBSKIP
  - See TBTOP
- MS commands referencing ISPF files can cause unpredictable results 141
- MSG (.MSG) control variable
  - See .MSG
- MSG(message-id) parameter
  - on DISPLAY 124
  - on LOG 139
  - on SETMSG 145
  - on TBDISPL 157
- MSG=value parameter
  - on assignment statement 215
- MVS
  - allocating libraries 66
  - batch environment 95
  - invocation of ISPF 87
  - invoking an ISPF application 5
  - library setup 65
  - PA keys
    - definition of 62
    - specifying options using parms 279
  - starting a dialog 12
  - TSO PCF 67
  - use of libraries 72



- name of variable too long for panel definition 226
- NAME parameter
  - on VER statement 221
- NAME(alt-name) parameter
  - on TBCLOSE 150
  - on TBSAVE 177
- NAME(member-name) parameter
  - on FTCLOSE 133
- name-list parameter
  - how to code on requests for services 108
  - on TBADD 147
  - on TBCREATE 153
  - on TBMOD 167
  - on TBPUT 171
  - on TBSARG 176
  - on TBSCAN 180

- on VCOPY 186
- on VDEFINE 188
- on VGET 192
- on VPUT 194
- on VREPLACE 196
- NAMENUM(namenum-name) parameter
  - on TBQUERY 173
- NAMES(name-list) parameter
  - on TBCREATE 153
- NAMES(var-name) parameter
  - on TBQUERY 173
- naming the command field via CMD
  - parameter 211
- NB parameter
  - on VER statement 220
- nested dialogs 51
- NEWAPPL(application-id) parameter
  - in )PROC section 230
  - on ISPSTART 89
  - on SELECT 140
- NEWCOPY parameter
  - on TBCLOSE 150
  - on TBSAVE 177
- NEWPOOL parameter
  - in )PROC section 230
  - on SELECT 140
- next message, set the display of
  - (SETMSG) 145
- NOBSCAN parameter
  - on VDEFINE 190
- NOCHECK parameter
  - in )PROC section 230
  - example 231
- NOFT parameter
  - on FTINCL 136
- NON parameter
  - with INTENS keyword 208
- non-ISPF displays, using REFRESH
  - after 50
- NONBLANK parameter
  - on VER statement 220
- NONDISPL parameter
  - on CONTROL 120
- nonreusable program used when split
  - screen mode use anticipated 68
- NOP action
  - specified via ZCTACT 54
- NOREPL parameter
  - on FTCLOSE 133
- NOWRITE parameter
  - on TBCREATE 153
  - on TBOpen 169
- NULLS parameter
  - with PAD keyword 209

- NUM parameter
  - on VER statement 220
- number of lines to scroll
  - See scrolling amount
- number parameter
  - on TBSKIP 182
- NUMBER(number) parameter
  - on TBSKIP 182
- numeric value parameter
  - how to code on requests for
    - services 108



- obtain a copy of variable (VCOPY) 186
- obtain copy of variable
  - from shared pool or profile
    - (VGET) 192
- obtain table information (TBQUERY) 173
- OFF parameter
  - with ATTN keyword 210
  - with CAPS keyword 208
  - with SKIP keyword 210
- ON parameter
  - with ATTN keyword 210
  - with CAPS keyword 208
  - with SKIP keyword 210
- online tutorial 16
- open a table (TBOpen) 169
- open skeleton files (FTOPEN) 137
- open table, create and (TBCREATE) 153
- operating system
  - passing commands to 46
  - operation in test mode 91
  - operation in trace mode 93
- OPT system variable 229
  - on tutorial panels 240
- OPT(option) parameter
  - on ISPSTART 89
  - on SELECT 140
- optional libraries
  - MVS 66
  - VM/SP 69
- options-list parameter
  - on VDEFINE 190
- order of panel definition sections 200
- order of search
  - variable pools 33
- output
  - protecting table 25
- OUTPUT parameter

with TYPE keyword 208  
overriding system commands by use of  
command tables 56

P

PA keys  
definition, VSE 63  
PA keys/definition  
MVS and VM/SP 62  
PAD keyword  
in panel )ATTR section 207, 209  
PAD(percentage) parameter  
on TBCLOSE 150  
on TBSAVE 177  
PAGE  
scrolling amount 52  
panel body section 210  
panel definition 199  
attribute section 199  
default characters 206  
statement format 207  
body section 199, 210  
sample 211  
statement format 210  
command field 211  
specification 211  
creation of 3, 19  
description of 3, 18  
design suggestions 202  
example of 19  
for help and tutorial panels 239  
format 200  
formatting guidelines 200  
initialization section 199, 213  
statement formats 214  
line 1  
content 201  
line 2 content 201  
line 3 content 201  
location of  
command field 200  
message field 200  
panel title 200  
scroll amount 200  
message field  
specification 211  
model section 199  
order of sections 200  
position of command field 201  
position of long message 201

processing 20, 227  
processing section  
statement formats 214  
sections of 199  
short message for TBDISPL  
operations 201  
special requirements 228  
specifying field type 208  
syntax rules 202  
tutorial and help panels 239  
used by TBDISPL 18  
panel display  
description of 7  
specification of 3  
panel processing considerations 227  
PANEL(panel-name) parameter  
in )PROC section 230  
on DISPLAY 124  
on ISPSTART 89  
on SELECT 140  
on TBDISPL 157  
PANELID system command 48  
parameters for service requests  
coding rules for iv, 108  
parameters specified as variables 107  
PARM(parameters) parameter  
in )PROC section 230  
on ISPSTART 89  
on SELECT 140  
parms option  
See also specifying options using  
parms  
description of 17  
instructions for use of 275  
terminal characteristics panel 276,  
277  
used for specifying ISPF parameter  
options 275  
parms option use  
MVS 279  
VM/SP 281  
VSE 282  
partitioning display screen  
See split screen mode  
parts of a dialog  
dialog components 7  
passing commands  
See command tables  
passing commands to a dialog  
function 57  
passing control  
from program-coded to command-coded  
function 9  
using the SELECT service 33

PASSTHRU action  
     specified via ZCTACT 54  
 PASSWORD(pswd-value) parameter  
     on BROWSE  
         MVS and VM/SP 113  
         VSE 116  
     on EDIT  
         MVS and VM/SP 126  
 PA1 key  
     MVS and VM/SP 62  
 PA1 key, VSE 63  
 PA2 key  
     MVS and VM/SP 62  
 PA2 key, VSE 63  
 PCF  
     TSO Programming Control Facility 67  
 percent (%) sign  
     beginning a command with 121, 141  
 percentage parameter  
     on TBCLOSE 150  
     on TBSAVE 177  
 permanent table 23  
 PF key-entered value  
     concatenated with PF key  
         restriction 47  
 PF keys 59  
     definition of functions for 60  
     definition panel 60  
     providing default settings 62  
     saving function definitions for 62  
     specifying  
         using parms option 275, 284  
     used for command entry 45, 46  
 PGM(program-name) parameter  
     in )PROC section 230  
     on ISPSTART 89  
     on SELECT 140  
 PICT parameter  
     on VER statement 220  
 PL/I  
     return codes from services  
         MVS and VM/SP 110  
         VSE 111  
 PL/I, compiler used with ISPF 1  
 PLIRETV/PLI built-in function  
     MVS and VM/SP 110  
 pointer, move current row  
     See TBBOTTOM  
     See TBSCAN  
     See TBSKIP  
     See TBTOP  
 POSITION(crp-name) parameter  
     on TBQUERY 173  
 positional parameters  
     call invocation 106  
     command invocation 104  
 PRESUME statement 103, 105  
 primary option menus 232  
 PRINT system command 49  
 PRINT-HI system command 49  
 processing  
     file tailoring services 28  
     light pen and cursor selected  
         fields 63  
         table services 23  
 processing a dialog 5  
 processing file tailoring skeletons  
     example 31  
 processing prior to panel display 213  
 processing tables  
     example use of table services 26  
 processing, panel 227  
 profile  
     See also application profile pool  
     definition of 33  
 PROFILE parameter  
     on VGET 192  
     on VPUT 194  
     on VPUT statement in panel )INIT and  
         )PROC sections 223  
 program access (PA) keys  
     MVS and VM/SP 62  
     VSE 63  
 program function keys 59  
 program function pool 36  
 program libraries  
     MVS 67  
     VM/SP 71  
 program linkage  
     VSE 3  
 program linking requirement for split  
     screen mode 68  
 program-name parameter  
     in panel )PROC section 230  
     on ISPSTART 87  
     on SELECT 140  
 program, copy a variable to (VCOPY) 186  
 pswd-value parameter  
     on BROWSE  
         MVS and VM/SP 113  
     on EDIT  
         MVS and VM/SP 126  
 put variables in shared pool or profile  
     (VPUT) 194

R

RANGE parameter  
 on VER statement 221

RCHANGE system command 47

read a table into virtual storage  
 (TBOPEN) 169

read row from table  
 TBBOTTOM 149  
 TBGET 165  
 TBSCAN 180  
 TBSKIP 182

read variable from shared pool or  
 profile (VGET) 192

recfm parameter  
 on BROWSE  
 VSE 116  
 on EDIT  
 VSE 129

recursive entry into dialog  
 functions 56

redisplaying contents of a screen using  
 PA Key  
 MVS and VM/SP 62  
 VSE 63

reentrant program used when split screen  
 mode use anticipated 68

reference of dialog variables by a  
 program function (VDEFINE) 188

referencing data in dialogs  
 See variable services description

REFRESH parameter  
 on CONTROL 120

relationship of function pools to dialog  
 functions 35

remove definition of variables from  
 function pool  
 VDELETE 191  
 VRESET 198

REPLACE parameter  
 on TBCREATE 153

replace variable in function pool  
 (VREPLACE) 196

REPLCOPY parameter  
 on TBCLOSE 150  
 on TBSAVE 177

representation of dialog variables 39

required libraries  
 MVS 65  
 VM/SP 68  
 VSE 73  
 VSE/AF 2.1 79

requirements  
 special for panel definition 228

reset of control variables 224

reset table variables to nulls  
 (TBVCLEAR) 185

reset variables (VRESET) 198

reshow key (PA2)  
 MVS and VM/SP 62  
 VSE 63

residency of tables 23

resource protection  
 table services 25

RESP (.RESP) control variable  
 See .RESP

RESTORE parameter  
 on CONTROL 120

restriction  
 on use of ISPF services 6

retrieve row from table  
 TBBOTTOM 149  
 TBGET 165  
 TBSCAN 180  
 TBSKIP 182

retrieve variables from shared pool or  
 profile (VGET) 192

return codes from services 109  
 MVS and VM/SP 110  
 VSE 111

RETURN mode  
 effect on error processing 109

RETURN parameter  
 on CONTROL 120

RETURN system command 47  
 distinguishing from END 51  
 processing 50

RETURN-CODE  
 COBOL built-in variable  
 MVS and VM/SP 110

RFIND system command 47

RIGHT parameter  
 with JUST keyword 208

RIGHT scroll command 52

RIGHT system command 48

row deletion (TBDELETE) 155

row display position  
 table display operations 201

row pointer, move  
 See TBBOTTOM  
 See TBSCAN  
 See TBSKIP  
 See TBTOP

row table services 26

row, determine existence (TBEXIST) 164

rownum-name parameter

- on TBQUERY 173
- rows of a table
  - content of 22
  - defining 23
- rules
  - coding parameters for service requests iv, 108
- running a dialog 5

S

- save and close table (TBCLOSE) 150
- SAVE parameter
  - on CONTROL 120
- save table (TBSAVE) 177
- SAVE(name-list) parameter
  - on TBADD 147
  - on TBMOD 167
  - on TBPOT 171
- SAVENAME(var-name) parameter
  - on TBBOTTOM 149
  - on TBGET 165
  - on TBSCAN 180
  - on TBSKIP 182
- saving PF key definitions, system variables for 62
- scrolling
  - commands to control 52
  - scroll amount 52
  - tutorial panels 52
- search
  - order of
    - for variable pools 33
- search a table (TBSCAN) 180
- search argument, specification of
  - TBSARG 175
  - TBSCAN 180
- SEL file tailoring control statement
  - See )SEL
- SEL system variable 229
  - See also ZSEL system variable
  - on tutorial panels 240
- SELECT action
  - specified via ZCTACT 54
- SELECT action command 55
- SELECT Service 14
  - creation of shared pools 38
  - description 14, 140
  - example 144
  - invocation 93
  - panel processing 229

- used to pass control within a dialog 33
- selecting an attention field 47
- selection panel
  - See menu
- serial parameter
  - on BROWSE
    - VSE 116
  - on EDIT
    - MVS and VM/SP 126
    - VSE 129
- serial/parameter
  - on BROWSE
    - MVS and VM/SP 113
- service interface routine, ISPLINK and ISPLNK 2
- service name parameter
  - coding in requests for services 108
- services
  - invocation 2
  - overview 17
  - to dialogs 1
  - to interactive applications 1
- services description
  - BROWSE 42
  - CONTROL 43
  - display 17
  - EDIT 42
  - file tailoring 28
  - LOG 43
  - SELECT 14
  - syntax for command and call statements iv, 103, 105, 112
  - table 22
  - variable 32
- SET file tailoring control statement
  - See )SET
- set next menu (ZPARENT) 233
- set processing modes (CONTROL) 120
- set row pointer
  - See TBBOTTOM
  - See TBSCAN
  - See TBSKIP
  - See TBTOP
- SETMSG service
  - description 18, 145
  - example 146
- SETVERB action
  - specified via ZCTACT 54
- shared minidisks, using, in VM 323
- SHARED parameter
  - on VGET 192
  - on VPUT 194

- on VPUT statement in panel )INIT and )PROC sections 223
- shared pool
  - creation of 38
  - definition of 33
  - description of 38
  - life of 38
  - used for communication between functions and panels 38
- shared pool variables
  - as affected by menu processing 35
- short message 20
- single name parameter
  - how to code on requests for services 108
- skel-name parameter
  - on FTINCL 136
- skeleton
  - description of 8
- skeleton definition
  - assigning a value to a variable 258
  - comment statement 258
  - description of 254
  - example 30, 259
  - format of 254
  - imbedding of 256
  - specifying table processing 257
  - types of records in 254
- skeleton formats 199
- SKIP keyword
  - in )ATTR section 210
  - in panel )ATTR section 207
- SKIP tutorial command 16, 239
- skipping display
  - See jump function
- SM parameter
  - on CONTROL 120
- SMSG parameter
  - on panel body section 211
- specify a table search argument
  - TBSARG 175
  - TBSCAN 180
- specify PF keys
  - alternate 287
  - primary 286
- specifying a table 23
- specifying ISPF parameter options
  - using parms option 275
- specifying log and list file defaults
  - using parms option 275
- specifying options using parms
  - MVS log and list defaults 279
  - PF keys 284
- terminal characteristics (option 0.1) 275
- VM/SP console, log, and list defaults 281
- VSE log and list defaults 282
- specifying terminal characteristics using parms option 275
- SPF
  - changes from 317
- SPLIT command 17
- SPLIT ENABLE,DISABLE parameters
  - on CONTROL 120
- split screen mode 16
  - command processing restriction 48
  - entering 17
  - program linking requirement 68
  - terminating 17
  - VSE restriction 93
- SPLIT system command 47
- stacking commands
  - for execution 46
  - HELP restriction 46
- START(line-number) parameter
  - on CONTROL 120
- starting a dialog 12
- starting ISPF 87
- steps in dialog development
  - MVS and VM/SP 72
  - VSE/AF 1.3.5 78
  - VSE/AF 2.1 85
- STOPAT system command 48
- storage-area parameter
  - in panel )PROC section 230
  - on ISPSTART 89
  - on SELECT 140
- storing variables from a panel in shared and profile pools (VPUT) 223
- subpools used by EDIT and BROWSE 42
- summary of changes from SPF 317
- summary of services syntax 301
- SWAP system command 47
- syntax rules
  - message definition 251, 253
  - panel definition 202
  - services requests (parameters) 108
  - services requests (service names and operands) iv
  - services summary 304
  - skeleton definitions 254
- sysno parameter
  - on BROWSE
    - VSE 116
  - on EDIT
    - VSE 129



- system commands 47
  - description 45
  - list of 47
  - overriding by use of command tables 56
  - PF key defaults 47
- system profile pool
  - See also application profile pool
  - accessible by 39
  - description of 39
  - order of search 39
- system variables 40
  - for saving PF key definitions 62
  - in shared pool 40
  - list of 41
  - used for communication between dialogs and ISPF 41
- system-defined default areas, panel 200

T

- table
  - avoiding concurrent update of 25
  - columns 22, 23
  - definition 22
  - description 7
  - example 23
  - input and output protection 25
  - input library
    - when same as output library 23
  - library access 23
  - permanent 23
  - processing 23
    - in conjunction with file tailoring processing 29
    - number of table limit 28
  - residency 23
  - rows 22, 23
  - size calculation 28
  - temporary 23
  - types 23
  - updates 23
    - when created or updated 8
- table display (TBDISPL) 156
- table display output
  - example 248, 250
- table display panel definition 243
  - attribute section 244
  - body section 244
  - example 247
  - example of multiple model lines 249

- initialization section 245
- message, location of 201
- model line 243
- model section 244
- scroll field, location of 201
- special requirements 228
- table display panel definition
  - short message area content 201
- table input and output protection 25
- table libraries
  - MVS 66
  - VM/SP 69
  - VSE 78, 84
- table save (TBSAVE) 177
- table search (TBSCAN) 180
- table services
  - description 21, 22
  - example 26
  - general services 24
  - resource protection 25
  - row services 26
  - syntax summary - calls 307
  - syntax summary - commands 304
  - used to modify displayed tables 289
- table update
  - avoiding concurrent 25
- table-name parameter
  - on TBADD 147
  - on TBBOTTOM 149
  - on TBCLOSE 150
  - on TBCREATE 153
  - on TBDELETE 155
  - on TBDISPL 157
  - on TBEND 162
  - on TBERASE 163
  - on TBEXIST 164
  - on TBGET 165
  - on TBMOD 167
  - on TBOPEN 169
  - on TBPUT 171
  - on TBQUERY 173
  - on TBSARG 176
  - on TBSAVE 177
  - on TBSCAN 180
  - on TBSKIP 182
  - on TBTOP 184
  - on TBVCLEAR 185
- table-row-number parameter
  - on TBDISPL 157
- TB
  - file tailoring control statement
    - See )TB
- TBADD service
  - description 147

- example 27, 148, 263
- TBADD services
  - used to modify displayed tables 261
- TBBOTTOM service
  - description 149
  - example 149
- TBCLOSE service
  - description 150
  - example 28, 152, 263, 290
- TBCREATE service
  - description 153
  - example 26, 154, 265
- TBDELETE service
  - description 155
  - example 155
- TBDISPL service
  - description 18, 156, 246
  - example 161, 289
- TBDISPL services
  - used to modify displayed tables 289
- TBEND service
  - description 162
  - example 162
- TBERASE service
  - description 163
  - example 163
- TBEXIST service
  - description 164
  - example 164
- TBGET service
  - description 165
  - example 166, 263
- TBMOD service
  - description 167
  - example 168
- TBOPEN service
  - description 169
  - example 170, 263, 290
- TBPUT service
  - description 171
  - example 290
  - examples 172
- TBQUERY service
  - description 173
  - example 174, 290
- TBSARG service
  - description 175
  - example 176
- TBSAVE service
  - description 177
  - example 179
- TBSCAN service
  - description 180
  - example 181
- TBSKIP service
  - description 182
  - example 183
- TBTOP service
  - description 184
  - example 184
- TBV CLEAR service
  - description 185
  - example 185
- TEMP parameter
  - on FTOPEN 137
- temporary table 23
- terminal characteristics
  - specifying
    - using parms option 275
- terminal display, specification of 3
- terminal keys 59
- terminating a dialog 93
- terminating display
  - See END system command
- terminating function or dialog
  - See END system command
  - See RETURN system command
- terminating ISPF 87, 93
- terminating TSO commands of CLIST using
  - PA Key 62
- TEST and TEXTX parameters
  - on ISPSTART 89
- test mode operation 91
- test value of variable during panel
  - processing 218
- testing dialog components 3
- TEXT keyboards
  - character translations 313
- TEXT parameter
  - with TYPE keyword 208
- TOP tutorial command 16, 239
- TRACE and TRACEX parameters
  - on ISPSTART 89
- trace mode operation 93
- TRAIL (.TRAIL) control variable
  - See .TRAIL
- TRANS parameter
  - on assignment statement 215
  - example 216, 217, 226, 230
  - example, nested 217
- translation of defined variable 40
- TRUNC parameter
  - on assignment statement 215
  - example 215, 217, 230
- truncation of defined variable 40
- TSO
  - batch environment 95
- TSO batch execution job 96

TSO Programming Control Facility 67  
TSO system command 48  
tutorial

- commands 16, 239
- ending of 16
- entry to 16
- invocation of 239
- sample hierarchy of panels 241
- sample panel 242
- use of 16, 239

TYPE keyword  
in panel )ATTR section 207, 208

U

UP scroll command 52  
UP system command 47  
UP system variable  
on tutorial panels 240  
UP tutorial command 16, 239  
update of tables  
avoiding concurrent 25  
update row in table  
TBMOD 167  
TBPUT 171  
update variables in shared pool or  
profile (VPUT) 194  
updating tables 23  
example use of table services 26  
use of commands, program keys, and light  
pen 45  
use of libraries  
MVS and VM/SP 72  
VSE/AF 1.3.5 78  
VSE/AF 2.1 85  
use of subpools by EDIT and BROWSE 42  
USER parameter  
on VDEFINE 189  
user-data parameter  
on VDEFINE 190  
using ISPF services 2  
restricted to ISPF environment 6  
using shared minidisks 323  
using the DISPLAY service 261  
using the parms option 275  
See also specifying options using  
parms  
using the TBDISPL service 289

V

validity checking user entered data on  
panel displays (use of the VER  
statement) 219  
value test of variable value during  
panel processing 218  
value-array parameter  
on VCOPY 186  
values parameter  
on VREPLACE 196  
var-name parameter  
on TBGET 165  
on TBQUERY 173  
on TBSCAN 180  
on TBSKIP 182  
variable  
parameter in panel definition  
    )INIT section 214  
    )PROC section 214  
variable parameter  
on assignment statement 215, 216  
on VDEFINE 188  
variable pools  
order of search 33  
variable services  
description 32  
summary 42  
syntax summary - calls 309  
syntax summary - commands 306  
variables  
access by functions written in a  
command procedure 35  
access by functions written in a  
programming language 36  
copy (VCOPY) 186  
creation of 35  
define in function pool  
(VDEFINE) 188  
defined type 36, 40  
definition of 32  
format of 39  
identical names 37  
implicit type 37  
in message definition 253  
maximum length of 32  
names of  
    passed as parameter to  
    services 104  
names too long for panel  
definition 226  
naming of 32

- on panels, restricted size 203
- order of accessing pools of 32
- pools of
  - order of search 33
- processing on menus 35
- remove definition of
  - from function pool (VDELETE) 191
  - from function pool (VRESET) 198
- replace in function pool (VREPLACE) 196
- reset 198
- retrieve from shared pool or profile (VGET) 192
- services description
  - See variable services description
- storing from a panel in shared and profile pools (VPUT) 223
- table extension 24
- update in shared pool or profile (VPUT) 194
- value test during panel processing 218
- variables, table
  - clearing to nulls using TBVCLEAR 185
  - KEY type 23
  - non-key type 23
- VCOPY service
  - description 186
  - example 187
  - used to access
    - system variables 40
- VDEFINE service
  - description 188
  - example 190
  - exit routine 188, 311
  - use of 36
- VDELETE service
  - description 191
  - example 191
- VER statement
  - in panel )INIT section 219
    - example 220, 221, 226
  - in panel )PROC section 219
- VER statement example 264
- verify variable content 219
- verlay phase structures n
  - t permitted in VSE 112
- VGET service
  - description 192
  - example 192
  - used to access
    - application profile pool 39
    - shared pool 33
    - system variables 40
- virtual machine communciation facility (VMCF), use of 44
- VL keyword
  - assembler language 106
- VM
  - shared minidisk support 323
- VM/SP
  - &PRESUME statement 105
  - allocating libraries 69
  - batch environment 98
  - batch execution job 98
  - invocation of ISPF 87
  - invoking and ISPF application 5
  - library setup 68
  - PA keys
    - definition of 62
  - PRESUME statement 103
  - restrictions on use of module files 72
  - specifying options using parms 281
  - starting a dialog 12
  - use of libraries 72
  - use of the virtual machine
    - communication facility (VMCF) 44
- VMCF, virtual machine communciation facility, use of 44
- VOLUME (serial) parameter
  - on EDIT
    - VSE 129
- VOLUME(serial) parameter
  - on BROWSE
    - MVS and VM/SP 113
    - VSE 116
  - on EDIT
    - MVS and VM/SP 126
- VPUT service
  - description 194
  - example 195
  - used to access
    - application profile pool 39
    - shared pool 33
- VPUT statement
  - example 223
  - in panel )INIT section 223
  - in panel )PROC section 223
- VREPLACE service
  - description 196
  - example 197
- VRESET service
  - description 198
  - example 198
- VSE
  - AF 1.3.5 library setup 73
  - batch environment 100

- batch execution job 100
- dialog abend intercept 95
- ENTRY statement required when link editing programs 112
- INCLUDE statements required when link editing programs 112
- invocation of ISPF 87
- invoking an ISPF application 5
- PA keys
  - definition of 63
- program linkage 3
- specify log and list defaults (Option 0.2) 282
- specifying options using parms 282
- split screen mode restriction 93
- starting a dialog 12
- VSE/AF 1.3.5
  - use of libraries 78
- VSE/AF 2.1 79
  - use of libraries 85

W

- write message to log file (LOG) 139
- WRITE parameter
  - on TBCREATE 153
  - on TBOpen 169
- writing dialogs, steps in
  - MVS and VM/SP 72
  - VSE/AF 1.3.5 78
  - VSE/AF 2.1 85

Z

- Z system variable 41
- Z variables used for field name placeholders 226
- ZAPPLID system variable 41
- ZCMD system variable 41
  - example 230
    - on tutorial panels 240
  - processing 229
    - blank 231
    - invalid option 231
- ZCMD, use of, versus other names for command field 201
- ZCONT system variable 41
  - on tutorial panels 240

- ZCTACT system variable 53
  - actions specified with 54
- ZCTDESC system variable 53
- ZCTTRUNC system variable 53
- ZCTVERB system variable 53
- ZDATE system variable 41
- ZDAY system variable 41
- ZERRHM system variable 41
- ZERRLM system variable 41
- ZERRMSG system variable 41
- ZERRSM system variable 41
- ZHINDEX system variable 41, 239
  - example
    - see figure on page 237
- ZHTOP system variable 41
  - example
    - see figure on page 237
- ZHTOP variable 239
- ZIND system variable 41
  - on tutorial panels 240
- ZJDATE system variable 41
- ZKEYS system variable 41
- ZLOGON system variable 41
- ZMONTH system variable 41
- ZPARENT system variable 41, 233
  - processing 233
- ZPF01-24 system variables 41
- ZPF01, ZPF02, ..., ZPF24 system variables 62
- ZPREFIX system variable 41
- ZPRIM system variable 41, 232
  - example
    - see figure on page 237
- ZSCBR system variable 41, 53
- ZSCED system variable 41, 53
- ZSCML system variable 41, 53
- ZSEL system variable 41
  - contains result of truncating ZCMD 229
  - example 230
    - on menus 229
    - on tutorial panels 240
  - parameters and keywords used with 230
- ZTDTOP system variable 41, 159
- ZTEMPF system variable 41, 137
- ZTERM system variable 41
- ZTIME system variable 41
- ZUP system variable 41
  - on tutorial panels 240
- ZUSER system variable 41
- ZVARS (.ZVARS) control variable
  - See .ZVARS
- ZVERB system variable 41

ZYEAR system variable 41



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.)



Reader's Comment Form

Cut or Fold Along Line

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 T46  
P. O. Box 60000  
Cary, North Carolina 27511



Fold and tape

Please Do Not Staple

Fold and tape

ISPF Dialog Management Services

Printed in U.S.A.

SC34-2088-2



Publication Number  
SC34-2088-2

File Number  
S370/4300-39

Program Number  
5668-960

Printed in  
U.S.A.

**IBM**<sup>®</sup>

SC34-2088-2

